Statistical learning for predictive targeting in online advertising

Fruergaard, Bjarne Ørum

Publication date:
2015

Document Version
Publisher's PDF, also known as Version of record

Ph.D. Thesis
in Machine Learning

DTU Compute
Department of Applied Mathematics and Computer Science

Statistical learning for predictive targeting in online advertising

Bjarne Ørum Fruergaard

Copenhagen 2014

University address:
DTU Compute
Department of Applied Mathematics and Computer Science
Technical University of Denmark

Richard Petersens Plads, Bygning 324
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Company address:
Adform

Hovedvagtsgade 6, 2. th.,
1103 København K, Denmark,
Phone +45 3535 7100
www.adform.com

PHD-2014-355
ISSN 0909-3192

Summary

The focus in this thesis is investigation of machine learning methods with applications in computational advertising. Computational advertising is the broad discipline of building systems which can reach audiences browsing the Internet with targeted advertisements. At the core of such systems, algorithms are needed for making decisions. It is in one such particular instance of computational advertising, namely in web banner advertising, that we investigate machine learning methods to assist and make decisions in order to optimize the placements of ads.

The industrial partner in this work is Adform, an international online advertising technology partner. This also means that the analyses and methods in this work are developed with particular use-cases within Adform in mind and thus need also to be applicable in Adform's technology stack. This implies extra thought on scalability and performance.

The particular use-case which is used as a benchmark for our results, is *click-through rate prediction*. In this task one aims to predict the probability that a user will click on an advertisement, based on attributes about the *user*, the *advertisement* the *context*, and other signals, such as *time*. This has its main application in real-time bidding ad exchanges, where each advertiser is given a chance to *place bids* for showing their ad *while the page loads*, and the winning bid gets to display their banner.

The contributions of this thesis entail application of a hybrid model of explicit and latent features for learning probabilities of clicks, which is a methodological extension of the current model in production at Adform. Our findings confirm that latent features can increase predictive performance in the setup of click-through rate prediction. They also reveal a tedious process for tuning the model for optimal performance.

We also present variations of Bayesian generative models for stochastic blockmodeling for inference of structure based on *browsing patterns*. Applying this structural information to improve click-through rate prediction becomes a two-step procedure; 1) learn user and URL *profiles* from browsing patterns, 2) use the profiles as additional features in a click-through rate prediction model. The assumption we implicitly make is reasonable: Users and URLs that are *grouped together* based on browsing patterns will have similar *responses* to ads, e.g., can be used as predictors of clicks. We report successful examples of applying this approach in practice.

Finally, we introduce the *multiple-networks stochastic blockmodel* (MNSBM), a model for efficient overlapping community detection in complex networks which can be assumed to be an aggregation of multiple block-structured subnetworks.

Resumé (Danish)

I denne PhD afhandling undersøger vi metoder indenfor maskinlæring med anvendelser indenfor datadrevet markedsføring ("computational advertising"). "Computational advertising" er en bred disciplin, der spænder over metoder, som bruges til målrettet markedsføring på internettet. Sådanne systemer bygger på algoritmer til automatisk at kunne træffe beslutninger. Det er særligt indenfor visning af web banner-reklamer, at vi i denne afhandling undersøger metoder i maskinlæring for at optimere beslutningsprocessen.

PhD projektet er et erhvervssamarbejde med Adform, som er leverandør af en digital markedsføringsplatform. Dette har indflydelse på de analyser og metoder, vi undersøger, da de bør kunne anvendes i Adforms systemer. Derfor har vi også ekstra fokus på skalérbare og højtydende metoder.

Den konkrete anvendelse, som bruges til at benchmarke vores resultater, er forudsigelse af klik-rater. I denne anvendelse er vi interesserede i at estimere sandsynligheden for, at en bruger vil klikke på en given reklame. Dette baseres på informationer om *brugeren*, *reklamen*, en *kontekst*, samt andre signaler, så som *tid*. Dette finder særligt anvendelse i online auktioner, hvor annoncører byder i realtid for at vinde retten til at vise netop deres reklame.

Bidragene i denne afhandling omfatter anvendelsen af en hybrid model, som indeholder både direkte og latente informationer, til estimering af klik-rater, og som er en udvidelse af den nuværende model i produktion hos Adform. Vores resultater bekræfter, at latente informationer kan læres ud fra data, og at de kan forbedre estimation af klik-rater.

Vi introducerer også variatoner af Bayesianske generative modeller til stokastisk blokmodellering af profiler baseret på besøgshistorikker. For at forbedre estimeringen af klik-rater, kan vi følge en procedure i to skridt; 1) først trænes profiler fra besøgshistorik, og 2) dernæst kan profilerne bruges som ekstra informationer i en model til estimering af klik-rater. Vi viser empirisk hvordan dette også bidrager til bedre estimering af klik-rater.

Til slut introducerer vi en ny model og metode til at detektere overlappende grupper fra observerede netværk. Modellen, som vi kalder "multiple-networks stochastic blockmodeling", fungerer under den antagelse, at det observerede netværk kan ses som en aggregering af mange delnetværk af simpel blokstruktur.

Preface

The PhD project was financed and conducted under the Industrial PhD Programme, which is managed by the Innovation Fund Denmark. The project is a collaboration between Adform ApS, the industrial partner and the employer of the PhD student, and the Section for Cognitive Systems in the department of Applied Mathematics and Computer Science (DTU Compute) at the Technical University of Denmark, where the PhD student, Bjarne Ørum Fruergaard, was enrolled at the ITMAN Graduate School. The PhD student has worked solely on the PhD project and has shared his time equally between the university and the industrial partner, in fulfillment of the Industrial PhD Programme guidelines.

University supervisor:
Prof. Lars Kai Hansen
lkai@dtu.dk

Danmarks Tekniske Universitet
Richard Petersens Plads,
Bygning 321, rum 012,
DK-2800 Lyngby, Danmark

Company supervisor:
PhD Jesper Urban
jesper.urban@adform.com

Adform ApS
Hovedvagtsgade 6, 2. th.
DK-1103 København K, Danmark

Copenhagen,

Bjarne Ørum Fruergaard

Acknowledgments

First of all, I would like to thank Adform and the Innovation Fund Denmark for financing of this Industrial PhD project.

I would like to thank also the Section for Cognitive Systems at DTU Compute for hosting me these three years and where I have had a relevant and scientifically rewarding course. Special thanks go to my university supervisor, Professor Lars Kai Hansen, for all the guidance I have received. A big thanks to all of my Cognitive Section colleagues and in particular I would like to thank my (some of them former) co-workers, Toke Jansen Hansen, Trine Julie Abrahamsen, Jens Brehm Nielsen, and Tue Herlau, both for their scientific input and help with my research, as well as for their friendship.

I would like to gratefully thank my employer Adform for making this project a reality. Big thanks also to my company supervisor and manager Jesper Urban, who has given me the valuable insights into the technical and business aspects needed for the project. I would also like to thank my co-workers in Adform for creating a nice and encouraging atmosphere and I look forward to working with you also in the future.

Finally, I would like to thank my family for the support they have given me throughout the project. In particular I would like to thank my loving wife, Malene, who has been the greatest of supports, in particularly when times have been hard and the hours have been long. I cannot say thanks enough for your support, Malene, and it is but a tiny gesture that I would like to dedicate this thesis to you.

Works included

Fruergaard, B. Ø.. Predicting clicks in online display advertising with latent features and side-information. *ArXiv e-prints arXiv:1411.7924 [stat.ML]*, November 2014.

Fruergaard, B. Ø., Hansen, T. J., and Hansen, L. K. Dimensionality reduction for click-through rate prediction: Dense versus sparse representation. *ArXiv e-prints arXiv:1311.6976 [stat.ML]*, November 2013.

Spotlight and poster presentation at the Probabilistic Models for Big Data workshop at Neural Information Processing Systems (NIPS), December 2013.

Fruergaard, B. Ø. and Hansen, L. K. Compact web browsing profiles for click-through rate prediction. In *Machine Learning for Signal Processing (MLSP), 2014 IEEE International Workshop on*, pages 1–6, September 2014.

Oral presentation at the IEEE International Workshop on Machine Learning for Signal Processing (MSLP), September 2014.

Fruergaard, B. Ø. and Herlau, T. Efficient inference of overlapping communities in complex networks. *ArXiv e-prints arXiv:1411.7864 [stat.ML]*, November 2014.

Contents

CHAPTER 1

Introduction

In this thesis we investigate machine learning methods with applications in computational advertising. Computational advertising is the broad discipline of building systems which can reach audiences browsing the Internet with targeted advertisements. At the core of such systems, algorithms are needed for making decisions. It is in one such particular instance of computational advertising, namely in web banner advertising, that we investigate machine learning methods to assist and make decisions about the *who*, *what*, *where*, and *when* in order to optimize the placements of ads.

The industrial partner in this work and key provider of the data that we analyze is Adform, an international online advertising technology platform. This also means that the analyses and methods in this work are developed with particular use-cases within Adform in mind and thus need also to be applicable within Adform's solutions. This implies extra thought on scalability and performance.

A particular use-case which is used as the benchmark platform for our results, is *click-through rate prediction*. In this task one aims to predict the probability that a user will click on an advertisement, based on attributes about the *user*, the *advertisement*, the *context*, and other signals, such as *time* (i.e., the who-what-where-when). This has its main application in real-time bidding ad exchanges, where each advertiser (or Adform acting on their behalf) are given a chance to *place bids* for showing their ad *while the page loads*, and the winning bid gets to display their banner.

The contributions of this thesis entail application of a hybrid model of explicit and latent features for learning probabilities of clicks, which is a methodological extension of the current model in production in Adform. Our findings confirm that latent features can increase predictive performance in the setup of click-through rate prediction, but they also reveal a tedious process for tuning the model for optimal performance.

We also present variations of Bayesian generative models for stochastic blockmodeling, which have a wide area of possible applications. Contrary to our work on the previously mentioned latent feature model, these generative models are applied on a different source of data, which is uninformed with respect to clicks. Particularly we model *browsing patterns*, represented as a bipartite graph of users and web sites where links represent visits and there is no embedded knowledge of clicks. Compared to click-through data, this stream of data has richer information about the *behavior* of users, both individually and collectively, which makes it easier to pick out struc-

ture. Applying this structural information to improve click-through rate prediction then becomes a two-step procedure; 1) learn user and URL *profiles* from browsing patterns, and 2) use the profiles as additional features in a click-through rate prediction model. The assumption we implicitly make is reasonable: Users and URLs that are *grouped together* based on their browsing patterns will have similar *responses* to ads, e.g., can be used as predictors of clicks. The models are non-parametric, meaning that model order is inferred automatically from data, and inference is done by Gibbs sampling with the most computationally demanding parts offloaded to GPUs for massively parallel computation. We report successful examples of applying this approach in practice.

Finally, we present a novel method we call *multiple-networks stochastic blockmodeling* (MNSBM) which models complex networks as the aggregation of multiple block-structured subnetworks. We demonstrate how this model improves link prediction on a number of real-world networks and discuss how inference can be scaled efficiently by running multiple GPU accelerated Gibbs samplers in parallel.

Thesis organization

The thesis is organized as follows. In the following we give a brief introduction to computational advertising, real-time bidding and the role that Adform plays. We then summarize the data sources from Adform that are relevant to our experiments. In Chapter 2 we introduce the theoretical foundations from machine learning that form the basis of the methodologies we investigate. This entails an introduction to probability theory, machine learning by optimization as well as Bayesian inference techniques. Chapter 3 focuses on a model for prediction of click-through rates from explicit features as well as an extension for learning latent features and we report results comparing the two. Chapter 4 takes a slightly different approach by splitting the inference problem in two; (i) learn latent variables and (ii) use those as predictors in a click-through rate prediction model. Intuitively, (i) can be thought of as unsupervised learning with respect to the task in (ii). Our focus is here on (i) and how we model large graphs of *browsing patterns* using Bayesian modeling techniques and GPU computing for scalable inference of stochastic blockmodels. We test different models and representations for this data and evaluate their utilities as predictors in (ii) by measuring the respective performances. In Chapter 5, we summarize and conclude on our work. Furthermore, the appendices C through F include the contributions that are part of this thesis.

1.1 Computational advertising

Computational advertising is a relatively new sub-discipline that builds on many current sciences, e.g., information retrieval, statistical modeling, machine learning, databases, economics, game theory, etc. It arises in the face of new technological possibilities with the growing popularity of the Internet as a publishing platform and

as a demand for automating the processes needed to monetize from online advertising. Its overall goal is to find the "best match" for a given *user* in a given *context* and a suitable *advertisement*. In the case of online web banners the context is usually a web site described by a URL, whereas in sponsored search advertising, the context is usually the keywords entered (query) by the user. In the latter case, a users' *intent* is more or less revealed through the query, whereas in the former, the user intent is usually not known, except that right now the user is looking at a particular URL. That means a challenge in web banner advertisement is to make the match between what users are looking at (historically) and trying to make connections to intent based on latent concepts of *interest*.

Computational advertising has been, and continues to be, a revolution from "the traditional" way of advertising through print media, billboards, TV, etc., and the industry around it has seen massive growth [43]. As opposed to the traditional advertising channels before the Internet, computational advertising sees billions of opportunities at relatively small cost, offers multitudes of different formats and creatives, including multimedia and interactivity, offers individualization, and is much more quantifiable [15]. A particular technology which supports this type of work flow is *real-time bidding* (RTB) auctions.

Real-time bidding

Since placement of web banner ads is the application in this thesis, we will introduce real-time bidding in this context, although it was popularized in sponsored search advertising by Google and Yahoo! [45].

The process of RTB begins when a user visits some website which is generating revenue by selling space for ads (*inventory*) on an *ad exchange*. The ad exchange then issues a request for bids including various sources of information about the user, context and ad slot to multiple advertisers, who then automatically respond bid prices. Then an auction takes place and the ad slot is sold to the advertiser with the highest bid. This entire process takes typically less than 100ms from the ad exchange first receives the request and is therefore called real-time bidding.

A possibility in RTB is thus to place bids which reflect some knowledge about the expected return on investment (ROI) based on the information that is given in the bid request. This is typically the role of *demand-side platforms* (DSP), which are platforms for the advertisers that take care of streamlining the information management and value estimation based on bid requests *from multiple ad exchanges* as well as offering the technology to be able to respond in less than 100ms. Adform offers a DSP for advertisers and can additionally leverage information from traditional ad serving (detailed in Section 1.3) in the process of value estimation. This leads to the main application of the work in this thesis in Adform.

1.2 Application in Adform

The primary area of application for the work in this thesis is for Adform's demand side platform. In Adform's DSP advertisers can set up advertising campaigns to buy advertising space via real-time bidding auctions on online ad exchanges. RTB thus puts the advertisers, the *demand side*, in control of who, what, where, when, and how much to buy, and opens up to much more specific and individualized advertising.

For real-time bidding a crucial component is estimation of *value*, which is needed in order to make reasonable bids. The majority of ad exchanges use some variant of generalized second-price or Vickrey-Clarke-Groves auctions, where the winner only pays the second highest bid, which is a means for incorporating an incentive for bidders to bid their true valuations (see e.g. Easley and Kleinberg [22, Chapter 15]).

Adform's DSP offers the advertisers to make the valuations and bid on their behalf. A popular valuation is for instance *cost per click* (CPC). Bidding with a value according to a target CPC, denoted CPC_t, and that is specified by the advertiser, takes place as follows,

$$\mathcal{B}(r) = CPC_t \cdot p(\text{click}|f(r)), \tag{1.1}$$

where r denotes a *request* (for bid), f is a function that extracts attributes from the request and yields a *feature vector*, and $\mathcal{B}$ is the bid price as a function of the request r. In practice Eq. (1.1) is a simplification of a more complicated setup, where for instance throttling is applied to ensure some minimum spend as well as making sure the campaign budget is not spent too quickly. Yet, the bid price is always proportional to Eq. (1.1) and hence the better we can model $p(\text{click}|f(r))$, the better the algorithm can optimize at what prices should it buy impressions. This methodology for buying impressions in RTB is also known as *performance based* pricing model [42].

1.3 An introduction to data

Adform as a technology provider for both publishers and advertisers collects data in many contexts and therefore we in this section give an introduction to those data which are relevant for the contents of this thesis.

Cookies For web technologies, cookies have a different meaning than their edible counterparts. In this thesis, when we refer to a cookie or more frequently a CookieId, we are referring to file that is created by a web browser and stored on disk. Among plenty of other purposes, Adform and/or the publisher that owns a web site can ask the users browser to store a cookie in which a CookieId is written down. The CookieId is created once with a unique identification number, when a specific browser on a specific computer enters a web site and does not already store a cookie for that web site. The user of a browser is in control of if and when they decide to delete the cookie, thus it may often appear as if a new user visits a web site, although that user (or their browser) has deleted an old cookie. While different technologies exist for

identifying users on the web, cookies are (still) by far the most wide-spread technology, and this is the only means of "tracking", although imprecise, that is assumed available for this thesis.

Ad serving Adform acts as a traditional *ad server* meaning that we take part in the delivery of ads, when Internet users visits a publisher with Adform ad serving. Generally speaking these are fixed-price deals between a publisher and usually one or more advertising agencies. Whenever an ad is shown to a user Adform writes down in its logs the user information, what page was accessed, which advertisement(s) was shown, as well as a number of other attributes, such as time, web browser, computer and OS attributes. Such a data record we refer to throughout this thesis as an *impression*. If an impression results in the user *clicking through* one of the adds that were served, then the entry in the log is marked as a *click*.

Real-time bidding When Adform takes part in RTB auctions on behalf of advertisers and agencies, the request from the ad exchange is logged. This request includes an identification of the user, which Adform is in many cases able to map to our own CookieId. Otherwise, only *if* an advertisement through Adform wins the auction, a CookieId is created and everything is logged in the same manner as in ad serving with the addition of Adform's own bid price as well as the winning price in the auction. If the exact URL being requested is available already when Adform receives the bid, this can be logged. Otherwise the URL is only logged when the auction is won and the ad is served by Adform.

Aggregated data Based on the data sources mentioned above, Adform can for instance aggregate over time a list of domains visited per CookieId, which is stored as a *cookie-profile* and can be used, e.g., as features in a supervised learning model.

CHAPTER 2

Background

In this chapter some key theoretical concepts that reoccur throughout the thesis are introduced. This entails an overall introduction of two machine learning principles; maximum posterior estimation and Bayesian inference. First a very brief introduction to probability theory.

2.1 Probability theory

As it is assumed the reader of this thesis should be already acquainted with the basic principles of probability theory, this section is not intended to teach a course in probability theory. Should the reader wish to catch up on the topic, [11] Section 1.2 is an excellent introduction.

Assuming a random variable X taking discrete values, the probability that X takes value x_i where $i = 1, \ldots, I$ is

$$p(X = x_i) = \frac{n_{x_i}}{N} \tag{2.1}$$

where n_{x_i} denotes the number of times event x_i is observed in a sample of size N, i.e., $n_{x_i} = \sum_{s=1}^{N} \delta_{x_i, X_s}$ with δ_{x_i, X_s} being the Kronecker delta taking the value 1 when $X_s = x_i$ and 0 otherwise. Strictly speaking Eq. (2.1) is the probability only in the limit $N \to \infty$; for finite N we can think of it as *maximum likelihood estimates*.

Let Y be another discrete random variable taking values $y_j = 1, \ldots, J$, we can refer to the probabilities of joint events $X = x_i$ *and* $Y = y_j$ as

$$p(X = x_i, Y = y_j) = \frac{n_{x_i, y_j}}{N} \tag{2.2}$$

where $n_{x_i, y_j} = \sum_{s=1}^{N} \delta_{x_i, X_s} \delta_{y_j, Y_s}$. We refer to Eq. (2.2) as the *joint probability*. If we sum $p(X = x_i, Y = y_j)$ over each discrete event y_j, we get the count n_{x_i} in the nominator, thus linking Eq. (2.2) to the *marginal probability* Eq. (2.1) by

$$p(X = x_i) = \sum_{j=1}^{J} p(X = x_i, Y = y_j) \tag{2.3}$$

which is the *sum rule* of probability.

If and when we consider only the data points where $X = x_i$ and we ask what is the probability of $Y = y_j$ in this subsample, we are *conditioning* on the variable X

and we refer to this as the *conditional probability* $p(Y = y_j|X = x_i)$, which is read as "what is the probability of $Y = y_j$ *given* $X = x_i$". This quantity is the fraction of points where $X = x_i$ and $Y = y_j$ over the number of events with $X = x_i$. This links the joint and marginal probabilities with the conditional probability by the following relation known as the *product rule* of probability

$$p(Y = y_j|X = x_i) = \frac{n_{x_i,y_j}}{n_{x_i}} = \frac{n_{x_i,y_j}/N}{n_{x_i}/N} \tag{2.4}$$

$$= \frac{p(X = x_i, Y = y_j)}{p(X = x_i)} \Leftrightarrow \tag{2.5}$$

$$p(X = x_i, Y = y_j) = p(Y = y_j|X = x_i)p(X = x_i). \tag{2.6}$$

Exploiting the symmetry $p(X = x_i, Y = y_j) = p(Y = y_j, X = x_i)$ and combining Eq. (2.5) and Eq. (2.6), we get *Bayes theorem*

$$p(Y = y_j|X = x_i) = \frac{p(X = x_i, Y = y_j)}{p(X = x_i)} = \frac{p(Y = y_j, X = x_i)}{p(X = x_i)} \tag{2.7}$$

$$= \frac{p(X = x_i|Y = y_j)p(Y = y_j)}{p(X = x_i)}. \tag{2.8}$$

Following the notation of [11], we will write $p(X)$ to denote a *probability distribution* over the possible outcomes for X or $p(x_i)$ as the equivalent of the more cumbersome notation $p(X = x_i)$. The interpretation should be clear from the context.

In this notation, we write Bayes theorem as

$$p(Y|X) = \frac{p(X|Y)p(Y)}{p(X)} \tag{2.9}$$

The denominator can be expressed as the sum over discrete events Y of the nominator

$$p(X) = \sum_Y p(X|Y)p(Y) \tag{2.10}$$

which we can think of as a normalization term, that ensures that the sum over discrete Y events in the left hand side of Eq. (2.9) always equals one.

So far we have considered discrete random variables, but these concepts also transfer to continuous valued random variables. If x and y are continuous random variables, we write the *probability density function* (pdf) as $p(x)$, i.e., the probability that x falls in the interval $(x, x + \Delta x)$ for $\Delta \to 0$. The sum and product rules become

$$p(x) = \int_y p(x, y)\mathrm{d}y \tag{2.11}$$

$$p(x, y) = p(y|x)p(x) \tag{2.12}$$

and Bayes theorem follows trivially

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)} \tag{2.13}$$

where again the denominator is the normalizer (and marginal)

$$p(x) = \int_y p(x|y)p(y)\mathrm{d}y. \tag{2.14}$$

A useful concept of probability theory is that of *independence*. If the joint probability decomposes into the product of marginals, then we call the events *independent*. I.e.,

$$p(x, y) \overset{x,y \text{ indep.}}{=} p(y)p(x). \tag{2.15}$$

Conditional independence is another useful property, which states that

$$p(x|y, z) \overset{x \text{ cond. indep. on } y}{=} p(x|z). \tag{2.16}$$

We say that x is conditionally independent on y given z. Combining the product rule and Eq. (2.16) gives us the alternative formulation

$$p(x, y|z) = p(x|y, z)p(y|z) \tag{2.17}$$

$$= p(x|z)p(y|z). \tag{2.18}$$

Assuming conditional independence sometimes allows us to break down complicated joint distributions conditionally on some variable into a series of simpler conditional distributions.

Bayesian probabilities

We now turn to slightly reformulating the probability theory of the previous section in a Bayesian setting. Our formulation so far has concentrated on the occurrence of either discrete or continuous valued events, e.g., probabilistically quantifying data. In machine learning we are interested in estimating the coefficients of some model or to make decisions about which model to use, in order to describe data in a meaningful way or to make the best and most informed decisions on future data. It turns out we can formulate our probability theory in a way that enables such decisions, which we refer to as *model inference*.

In the following we assume an observed dataset $\mathcal{D} = \{d_1, \ldots, d_N\}$ and we wish to fit a model with coefficients $\boldsymbol{\phi}$ to this data. We specify a distribution $p(\boldsymbol{\phi})$ called the *prior distribution* which captures our beliefs about the model coefficients prior to observing any data. We can then formulate Bayes theorem

$$p(\boldsymbol{\phi}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{\phi})p(\boldsymbol{\phi})}{p(\mathcal{D})}. \tag{2.19}$$

The left hand side $p(\boldsymbol{\phi}|\mathcal{D})$ is called the *posterior distribution* of the model parameters, i.e., this quantifies the uncertainties in the model parameters after (alas *posterior*) observing the data.

The distribution $p(\mathcal{D}|\boldsymbol{\phi})$ on the right hand side of Eq. (2.19) is called the *likelihood function*. The likelihood function captures the probability of observing the data $\mathcal{D}$ given the model parameters $\boldsymbol{\phi}$. The denominator on the right hand side ensures that the posterior distribution on the left is a valid probability density function and integrates to one. As in the previous section, we can express this in terms of an integral w.r.t. the parameters of the nominator in Eq. (2.19)

$$p(\mathcal{D}) = \int_{\boldsymbol{\phi}} p(\mathcal{D}|\boldsymbol{\phi})p(\boldsymbol{\phi})\mathrm{d}\boldsymbol{\phi}. \tag{2.20}$$

Many machine learning methods are based on *maximum likelihood estimation* (MLE) for finding the unknown parameters $\boldsymbol{\phi}$. As the name suggests, in MLE we pick the model parameters $\boldsymbol{\phi}$ that maximize the likelihood term $p(\mathcal{D}|\boldsymbol{\phi})$. This is usually done by taking the negative log of the likelihood, corresponding to what is known as a *loss function*, and the problem becomes that of minimizing the loss.

MLE is an example of *frequentist* modeling. In frequentist methods, the parameters $\boldsymbol{\phi}$ are assumed to have fixed but unknown values, i.e., the parameters are not assumed stochastic, whereas in Bayesian methods we infer the posterior distribution over parameters $p(\boldsymbol{\phi}|\mathcal{D})$ or we are able to draw samples from it. While the latter naturally enables error bars on the parameters given the observed data, in frequentist methods we can in principle only derive confidence intervals over parameters by (re)sampling the data, e.g., using the *bootstrap* [23]. Hence the interpretations and conclusions that we can draw from models differ based on whether we do MLE or Bayesian inference.

Predictive models

Whether we do frequentist or Bayesian inference, for the scope of this thesis we are interested in making prediction for unseen data. Generally, we will have a dataset where each data point has some *features*, which for simplicity we will assume can be represented in a vector $\boldsymbol{x}$, and a target variable or label y. The task is then to learn a model with a feature vector as input which is able to predict the target variable.

The first step in predictive modeling is to specify a model of how the features relate to the target. Typically we choose a probability distribution which is parameterized by the features as well as other parameters $\boldsymbol{\phi}$

$$p(y|\boldsymbol{x}, \boldsymbol{\phi}) = \mathbb{P}\mathrm{r}(y|\boldsymbol{x}, \boldsymbol{\phi}), \tag{2.21}$$

where $\mathbb{P}\mathrm{r}$ represents a concrete probability distribution (e.g., Gaussian) suitable for the target variable.

We then train a predictive model by splitting the dataset into separate training and testing data and learning the parameters from the training data alone. The

training features we denote as a matrix $\mathbf{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$ and the training targets as a vector $\mathbf{y} = (y_1, \ldots, y_N)^T$, i.e., a data point $d_n \in \mathcal{D}$ is the tuple $(\boldsymbol{x}_n, y_n)$. Assuming the individual data points are *independent and identically distributed* (iid.), we can write the likelihood as

$$p(\mathbf{y}|\mathbf{X}, \boldsymbol{\phi}) = \prod_{n=1}^{N} \mathbb{P}\mathrm{r}(y_n|\boldsymbol{x}_n, \boldsymbol{\phi}). \tag{2.22}$$

In MLE we thus proceed by maximizing the likelihood Eq. (2.22) directly, thereby obtaining the parameter vector $\boldsymbol{\phi}_{ML}$. For making predictions on new a new data point $\boldsymbol{x}^*$ we just plug in $\boldsymbol{\phi}_{ML}$ in Eq. (2.21) and predict using

$$p(y|\boldsymbol{x}^*, \mathbf{X}, \mathbf{y}, \boldsymbol{\phi}_{ML}) = \mathbb{P}\mathrm{r}(y|\boldsymbol{x}^*, \boldsymbol{\phi}_{ML}). \tag{2.23}$$

For Bayesian inference we also specify a prior $p(\boldsymbol{\phi}|\alpha)$, where for simplicity we assume α is a single, known parameter of the prior distribution. We call α (and any parameter of the priors) a *hyper-parameter*. Using the sum and product rules of probability, we can write out the *predictive distribution*

$$p(y|\boldsymbol{x}^*, \mathbf{X}, \mathbf{y}) = \int_{\boldsymbol{\phi}} p(y|\boldsymbol{x}^*, \mathbf{X}, \mathbf{y}, \boldsymbol{\phi}) p(\boldsymbol{\phi}|\mathbf{X}, \mathbf{y}) \mathrm{d}\boldsymbol{\phi} \tag{2.24}$$

$$(y \text{ iid.}) \quad = \int_{\boldsymbol{\phi}} p(y|\boldsymbol{x}^*, \boldsymbol{\phi}) p(\boldsymbol{\phi}|\mathbf{X}, \mathbf{y}) \mathrm{d}\boldsymbol{\phi}, \tag{2.25}$$

where the dependency on α is left out only to keep the notation simpler. On the right hand side $p(y|\boldsymbol{x}^*, \boldsymbol{\phi})$ is just Eq. (2.21) and $p(\boldsymbol{\phi}|\mathbf{X}, \mathbf{y})$ is the posterior of the parameters given training data. In special cases the integral Eq. (2.25) has an analytical solution, but if not, we can approximate using a finite sample

$$p(y|\boldsymbol{x}^*, \mathbf{X}, \mathbf{y}) \simeq \frac{1}{L} \sum_{l=1}^{L} p(y|\boldsymbol{x}^*, \boldsymbol{\phi}^{(l)}), \tag{2.26}$$

provided the samples $\boldsymbol{\phi}^{(l)}$ are drawn independently from $p(\boldsymbol{\phi}|\mathbf{X}, \mathbf{y})$. This is also known as *Monte Carlo* simulation. We will come back to Bayesian inference of $p(\boldsymbol{\phi}|\mathbf{X}, \mathbf{y})$ as well as sampling in a later section.

Instead of finding the posterior distribution and being able to sample from it, we can use a frequentist approach which improves on MLE by incorporating priors as well as the likelihood. From Eq. (2.19), we see that

$$p(\boldsymbol{\phi}|\mathbf{X}, \mathbf{y}) \propto p(\mathbf{X}, \mathbf{y}|\boldsymbol{\phi}) p(\boldsymbol{\phi}|\alpha), \tag{2.27}$$

i.e., the posterior distribution is proportional to the likelihood times the prior. In the same manner as MLE, we can optimize for the value of $\boldsymbol{\phi}$ which maximizes the posterior, instead of the likelihood only. This is known as *maximum posteriori* estimation (MAP). So instead of obtaining the full posterior $p(\boldsymbol{\phi}|\mathbf{X}, \mathbf{y})$ (or samples from it) we get the single parameter vector $\boldsymbol{\phi}_{MAP}$ which maximizes Eq. (2.27).

2.2 Machine learning by optimization

For a lot of popular machine learning methods these can be cast as special cases of MLE or MAP. For example, the straight forward formulation of logistic regression corresponds to MLE with a Bernoulli likelihood parameterized by a non-linear function over a linear model of the input features. For regularized logistic regression a penalty on the coefficients is added to the objective function being minimized and in many cases this penalty can be directly derived from a prior in a Bayesian formulation, thus making this a MAP approach.

In this section we introduce the techniques that are general for optimization problems, but which are applied in this thesis to obtain MAP solutions.

As mentioned in the previous section in MAP we focus on the posterior $p(\boldsymbol{\phi}|\mathcal{D})$ and wish to pick the parameters of our model $\boldsymbol{\phi}$ such that the posterior is maximized. Since we are maximizing, the normalization term of Bayes theorem is disregarded as a constant and the optimization problem becomes

$$\underset{\boldsymbol{\phi}}{\operatorname{argmax}} \quad p(\mathcal{D}|\boldsymbol{\phi})p(\boldsymbol{\phi}|\alpha) \tag{2.28}$$

We assume that data is iid. and we can write

$$\underset{\boldsymbol{\phi}}{\operatorname{argmax}} \quad p(\boldsymbol{\phi}|\alpha) \prod_{d\in\mathcal{D}} p(d|\boldsymbol{\phi}) \tag{2.29}$$

Since working with a product of possibly very small probabilities is impractical and because numerical optimization is often formulated as a minimization problem, we take the negative logarithm of Eq. (2.29) and arrive at

$$\underset{\boldsymbol{\phi}}{\operatorname{argmin}} \quad -\log p(\boldsymbol{\phi}|\alpha) - \sum_{d\in\mathcal{D}} \log p(d|\boldsymbol{\phi}) \tag{2.30}$$

Since the negative logarithm is a monotonically decreasing function, the solution of Eq. (2.30) is equivalent to Eq. (2.29).

Eq. (2.30) is a classic example of an optimization problem where we have a data-dependent *loss function*, $-\sum_d \log p(d|\boldsymbol{\phi})$, and a *regularization* (or penalty) term, $-\log p(\boldsymbol{\phi}|\alpha)$, which is a function of the model parameters only. For a general regularized optimization problem we write

$$\underset{\boldsymbol{\phi}}{\operatorname{argmin}} \quad \Omega(\boldsymbol{\phi}, \alpha) + L(\mathcal{D}, \boldsymbol{\phi}), \tag{2.31}$$

where in this case

$$L(\mathcal{D}, \boldsymbol{\phi}) = -\sum_{d\in\mathcal{D}} \log p(d|\boldsymbol{\phi}) \tag{2.32}$$

$$\Omega(\boldsymbol{\phi}, \alpha) = -\log p(\boldsymbol{\phi}|\alpha) \tag{2.33}$$

and $L(\cdot)$ is the loss function and $\Omega(\cdot)$ is the regularizer. In the more general framework of numerical optimization, the loss function could be any real-valued lower-bounded function of $\boldsymbol{\phi}$ and $\mathcal{D}$ that we would like to minimize subject to $\boldsymbol{\phi}$. In this thesis, the only loss we consider is the *negative log-likelihood* given in Eq. (2.32).

In the following we introduce two kinds of regularizers, namely ℓ^2 and ℓ^1 regularization.

Regularization functions and MAP

Assume we have an optimization problem as in Eq. (2.30). Let $k = 1, ..., K$ index the parameter vector $\boldsymbol{\phi}$. We now choose a zero-mean Gaussian prior with variance $\sigma^2 = \frac{1}{2\lambda}$

$$p(\boldsymbol{\phi}|\lambda) = \prod_k \mathcal{N}(\phi_k|0, \tfrac{1}{2\lambda}) \tag{2.34}$$

$$= \left(\tfrac{\lambda}{\pi}\right)^{K/2} \prod_k \exp(-\lambda\phi_k^2) \tag{2.35}$$

Then taking the negative log

$$-\log(\prod_k \mathcal{N}(\phi_k|0, \tfrac{1}{2\lambda})) = \lambda \sum_k \phi_k^2 - \tfrac{K}{2} \log\left(\tfrac{\lambda}{\pi}\right) \tag{2.36}$$

In particular, we see that $\sum_k \phi_k^2 = ||\boldsymbol{\phi}||_2^2$, where $||\cdot||_2$ denotes the ℓ^2-norm, also known as the Euclidean norm of a vector. Now inserting Eq. (2.36) back into the optimization problem Eq. (2.30), we see that the last term, $-\frac{K}{2} \log\left(\frac{\lambda}{\pi}\right)$, can be dropped, since it is constant w.r.t. $\boldsymbol{\phi}$, and we get

$$\underset{\boldsymbol{\phi}}{\operatorname{argmin}} \quad \lambda||\boldsymbol{\phi}||_2^2 - \sum_{d \in \mathcal{D}} \log p(d|\boldsymbol{\phi}), \tag{2.37}$$

This regularizer is called ℓ^2 regularization. The formulation above draws parallels to the *Lagrangian function*, $\mathcal{L}(\boldsymbol{\phi}, \lambda)$ where λ plays the role of a *Lagrange multiplier* ([62] Chapter 12). In constrained optimization, the above corresponds to a problem where the negative log-likelihood is optimized subject to

$$||\boldsymbol{\phi}||_2^2 \leqslant \eta \tag{2.38}$$

for an appropriate value of the parameter η. This connection is useful in a moment, when we geometrically interpret regularization.

The ℓ^2-regularizer encourages weights to be pushed towards the origin in parameter space, unless otherwise supported in data, and it is therefore also known by the name of *weight decay*.

We now assume a different prior on $\boldsymbol{\phi}$, namely a zero-mean Laplace distribution with scale parameter $b = \frac{1}{\lambda}$

$$p(\boldsymbol{\phi}|\lambda) = \prod_k \text{Laplace}(\phi_k|0, \tfrac{1}{\lambda}) \tag{2.39}$$

$$= \left(\tfrac{\lambda}{2}\right)^K \prod_k \exp(-\lambda|\phi_k|) \tag{2.40}$$

where $|\cdot|$ denotes the absolute value. Again, taking the negative log

$$-\log(\prod_k \text{Laplace}(\phi_k|0, \tfrac{1}{\lambda})) = \lambda \sum_k |\phi_k| - K \log\left(\tfrac{\lambda}{2}\right), \tag{2.41}$$

We see that $\sum_k |\phi_k|$ is the ℓ^1-norm, $||\boldsymbol{\phi}||_1$, and again insert into Eq. (2.30) and eliminate the constant term, $-K \log\left(\frac{\lambda}{2}\right)$. Hence, we get

$$\underset{\boldsymbol{\phi}}{\text{argmin}} \quad \lambda||\boldsymbol{\phi}||_1 - \sum_{d\in\mathcal{D}} \log p(d|\boldsymbol{\phi}) \tag{2.42}$$

This regularizer is called ℓ^1 regularization.

The ℓ^1 regularizer, like the ℓ^2, shrinks parameters towards zero. In the context of constrained optimization, the above minimization problem can be formulated as minimizing the negative log-likelihood subject to

$$||\boldsymbol{\phi}||_1 \leqslant \eta \tag{2.43}$$

which we will use in a moment.

ℓ^1 is also known as the *lasso* operator, short for *least absolute shrinkage and selection operator*, and contrary to ℓ^2 it promotes sparsity in the parameter vector (hence *selection operator*).

In Fig. 2.1 we show the contours of a quadratic function and the constraint regions for ℓ_2 and ℓ_1 regularizers that arrive from Eq. (2.38) and 2.43, respectively. The solutions marked in the figure are the minima of the objective function defined by the quadratic plus the regularizers. In the case of ℓ_1 we see that the solution only has one non-zero component, i.e., this solution is sparse. This geometric interpretation shows us the intuition of why parameters are shrunk towards the origin and in particularly, it shows why solutions tend to be sparse using ℓ_1 regularization. With ℓ_1 regularization, the constraint region is the simplex which due to its "angled" shape along the axes constrains the solution towards zeroing out components; an effect which is amplified as the number of dimensions increase.

Controlling overfitting

In the previous section we have seen how the ℓ_2 and ℓ_1 regularizers correspond to MAP estimation of a likelihood and particular choices of priors. Often in practice

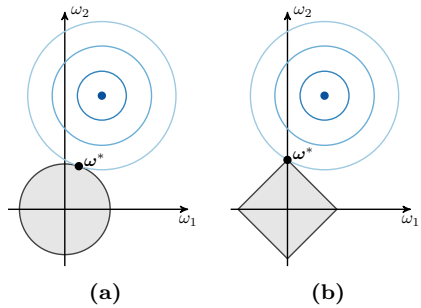


Figure 2.1: Plots of the contours (darker blue means lower function value) of a quadratic function in the two parameters $\boldsymbol{\omega} = (\omega_1, \omega_2)^T$ and the constraint regions defined by the regularizers **(a)** ℓ_2 and **(b)** ℓ_1. The minimizer in each of the two cases ($\boldsymbol{\omega}^*$) is found as the point inside or at the boundary of the constraint function, where the quadratic function is lowest.

this connection is not attributed much attention. Instead, regularization is mostly added in order to *control overfitting*.

Overfitting is a phenomenon particularly expressed in MLE, when the model that is trained on some dataset $\mathcal{D}$ is used for predicting unseen observations. In essence, MLE fits also the noise in the training data and this yields poor performance when used for predictions. This "fit" of noise in MLE can be controlled by adding regularization and is connected to the *bias-variance trade-off*, covered in great detail in Bishop [11, Section 3.2]. However, when adding regularization the question arises how to select an appropriate regularization strength λ. Ideally this would be the one which minimizes the loss for all conceivable *test sets* on average, but that is rarely possible in practice.

Instead we can use *cross-validation*, where an independent test set is used to benchmark different values of λ. Provided enough independent tests, this allows us to pick a good regularization strength.

Other alternatives to cross-validation exists, e.g., maximizing the *evidence function* (Bishop [11, Section 3.5]). In fully Bayesian settings where predictions are based on averages where the parameters are marginalized out, overfitting is virtually nonexisting. As demonstrated in Hansen [35], even in cases where priors as well as hyperpriors are introduced to hypothesize over unknown (hyper-)parameter distributions, Bayesian averaging can manage to trade-off bias versus variance optimally.

For hyper-parameter tuning in Adform, we generally perform cross-validation in a manner which reflects the sequential flow of data; train on a sample of $N-1$ days in the past with varying hyper-parameters and report performance on the N^{th} day. Continuing these experiments over a number of, say K, days, we can average the test set performances in a similar manner to k-fold cross-validation and pick the

hyper-parameters with the best generalization over test sets independent also in time.

Numerical optimization

We now return to the general formulation of a regularized optimization problem

$$\underset{\boldsymbol{\phi}}{\operatorname{argmin}} \quad f(\boldsymbol{\phi}) \equiv \Omega(\boldsymbol{\phi}, \alpha) + L(\mathcal{D}, \boldsymbol{\phi}) \tag{2.44}$$

We will refer to the function that we are minimizing, in this case $\Omega(\boldsymbol{\phi}, \alpha) + L(\mathcal{D}, \boldsymbol{\phi})$, as the *objective function* $f(\boldsymbol{\phi})$. In this section we introduce the techniques that we apply in order to solve (or approximate) Eq. (2.44).

For some choices of $L(\cdot)$ and $\Omega(\cdot)$ we may be able to solve Eq. (2.44) analytically. In particular when the objective function is convex and differentiable everywhere, we can differentiate it and equate it to zero to get the solution. A classic example is ℓ^2 regularized least-squares, which has a closed-form solution (see e.g., [37] Section 3.4.1).

For the loss and regularization choices we make use of later in this thesis there does not exist closed-form solutions. In this case we apply sequential algorithms which are the focus of this section. Unless otherwise stated, for the proofs of any claims in this section, we refer to [62], which covers numerical optimization in great detail.

For the moment, assume that the objective function in Eq. (2.44) is differentiable in $\boldsymbol{\phi}$ everywhere, then the gradient $\nabla_{\boldsymbol{\phi}} f(\boldsymbol{\phi})$ exists. In the following, for simplicity we write ∇f_t for the gradient of f w.r.t. $\boldsymbol{\phi}_t$, where the subscript denotes an iterate at time t. We now introduce an iterative procedure where in each iteration, we solve

$$\min_{\eta > 0} f(\boldsymbol{\phi}_t + \eta \boldsymbol{p}_t) \tag{2.45}$$

for a search direction $\boldsymbol{p}_t$; then assigning the next iterate $\boldsymbol{\phi}_{(t+1)} = \boldsymbol{\phi}_t + \eta \nabla f_{(t)}$. In numerical optimization this is known as a *line search* algorithm, which in each iteration searches for the step length η along direction $\boldsymbol{p}_t$ with the highest benefit.

Search direction

The search direction $\boldsymbol{p}_t$ can be any direction which decreases f ($\boldsymbol{p}_t$ is called a *descent direction*). Using the negative gradient $-\nabla f_t$ as the search direction is a natural choice, since it is the direction of *steepest descent*, i.e., along which $f(\boldsymbol{\phi}_t)$ decreases most rapidly. Steepest descent however, does not take into account the curvature in f around a solution, characterized by the Hessian $\nabla^2 f_t$, which makes it terribly ineffective for nonlinear problems.

An important descent direction is *Newtons direction*. Consider a second-order Taylor series expansion of $f(\boldsymbol{\phi}_t + \boldsymbol{p})$,

$$f(\boldsymbol{\phi}_t + \boldsymbol{p}) \approx f_t + \boldsymbol{p}^T \nabla f_t + \tfrac{1}{2} \boldsymbol{p}^T \nabla^2 f_t \boldsymbol{p} \equiv m_t(\boldsymbol{p}). \tag{2.46}$$

If we now assume that ∇^2 is positive definite (i.e., it is invertible), Newtons directions is the direction $\boldsymbol{p}$ which minimizes $m_t(\boldsymbol{p})$. Then taking the derivative of $m_t(\boldsymbol{p})$ and equating it to zero, yields Newtons direction

$$\boldsymbol{p}_t^{\text{N}} = -\nabla^2 f_t^{-1} \nabla f_t. \tag{2.47}$$

When $\nabla^2 f_t$ is positive definite, Newtons direction is a descent direction that can be used in a line search method. I.e., consider Eq. (2.47), that can be rewritten

$$\boldsymbol{p}_t^{\text{N}} = -\nabla^2 f_t^{-1} \nabla f_t \Leftrightarrow \tag{2.48}$$

$$\nabla f_t = -\nabla^2 f_t \boldsymbol{p}_t^{\text{N}} \Leftrightarrow \tag{2.49}$$

$$\nabla f_t^T \boldsymbol{p}_t^{\text{N}} = -\boldsymbol{p}_t^{\text{N}T} \nabla^2 f_t \boldsymbol{p}_t^{\text{N}} \leq -\sigma_t ||\boldsymbol{p}_t^{\text{N}}||_2^2 \tag{2.50}$$

where the latter inequality holds for some $\sigma_t \geq 0$ and follows from $\nabla^2 f_t$ being positive definite. Using $\boldsymbol{p}_t^{\text{N}}$ as the search direction in Eq. (2.45) is known in optimization as *Newtons method*.

It can be shown ([62] Section 3.3), that with proper assumptions on f and assuming that the line searches are exact, the convergence rate of steepest descent is *linear*, whereas it is *quadratic* for Newtons method. Convergence rates describe properties about the quotient of the errors on a sequence $\{\boldsymbol{\phi}_t\}$ in $\mathbb{R}^n$ converging to the local minimizer $\boldsymbol{\phi}^*$. Linear convergence is if there exists a constant $r \in (0, 1)$ such that

$$\frac{||\boldsymbol{\phi}_{t+1} - \boldsymbol{\phi}^*||_2}{||\boldsymbol{\phi}_t - \boldsymbol{\phi}^*||_2} \leq r, \quad \text{for all } t \text{ sufficiently large.} \tag{2.51}$$

I.e., the distance to the solution $\boldsymbol{\phi}^*$ decreases at least by at least a constant each iteration.

Quadratic convergence is obtained when there exists a constant factor $M > 0$ such that

$$\frac{||\boldsymbol{\phi}_{t+1} - \boldsymbol{\phi}^*||_2}{||\boldsymbol{\phi}_t - \boldsymbol{\phi}^*||_2^2} \leq M, \quad \text{for all } t \text{ sufficiently large.} \tag{2.52}$$

Consequently, we prefer quadratic convergence over linear. However, as we point out in the following, the exact Newtons method is not feasible for large-scale problems.

For problems with large feature spaces (K is large), as we will be concerned with in this thesis, the inverse Hessian $\nabla^2 f_t^{-1}$ required in Eq. (2.47) is infeasible to use for a number of reasons. Recall that the Hessian is a $K \times K$ symmetric matrix of the second partial derivatives of f

$$\nabla^2 f(\boldsymbol{\phi}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_K} \\ \vdots & & \vdots \\ \frac{\partial^2 f}{\partial x_K \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_K^2} \end{bmatrix} \tag{2.53}$$

First of all this needs to be invertible (positive definite), although even if it is not, there are still ways to approximate $\nabla^2 f_t^{-1}$. Second, calculating this from scratch in each iteration can be very costly.

This has led to the development of *quasi-Newton* methods, in which instead of explicitly calculating the Hessian, this is approximated by a matrix $\boldsymbol{B}_t$, which is updated after each iteration to take the newest gradient information into account in the approximation. The *BFGS formula*, named after its inventors, Broyden, Fletcher, Goldfarb, and Shanno, is amongst the most popular of such techniques. This method is described at length in [62] Chapter 8 and we will not delve into it further here.

Yet even with the BFGS approximation, the $K \times K$ matrix approximating the Hessian (or its inverse $\nabla^2 f_t^{-1}$ required in 2.47 and makes for a more efficient implementation) will be generally dense, so storage and manipulating it becomes intractable for large K. Hence a further advancement is L-BFGS, "L" standing for limited memory has been introduced, which is suitable for large-scale unconstrained optimization and builds on BFGS. Again, for a detailed explanation of L-BFGS, we refer the reader to [62] Chapter 9. Here we only give a motivation of its workings: In BFGS, $\boldsymbol{B}_t^{-1}$ is updated each iteration based on the previous and the new iterates, $\boldsymbol{\phi}_t$ and $\boldsymbol{\phi}_{t+1}$, as well as the previous and the new gradients, ∇f_t and ∇f_{t+1}. In L-BFGS the last M (called the *memory* and which the user decides) iterates and gradients are stored and then the multiplication $\tilde{B}_t^{-1} f_t$ for the approximated Newton direction in Eq. (2.47) can be computed as a sequence of inner products and vectors summations using f_t, the previous M iterates and gradients only, thus avoiding ever explicitly representing the *limited memory* version of $\tilde{B}_t^{-1}$. In other words, L-BFGS maintains only the curvature information from the most recent M points and thus avoids storing the otherwise unmanageably large full (estimated) Hessian matrix.

For quasi-Newton based methods, such as L-BFGS, there is a trade-off in terms of convergence speed compared to the quadratic convergence of the pure Newtons method. The convergence rate of the BFGS methods is in general *superlinear* ([62] Chapter 8), meaning

$$\lim_{t \leftarrow \infty} \frac{||\boldsymbol{\phi}_{t+1} - \boldsymbol{\phi}^*||_2}{||\boldsymbol{\phi}_t - \boldsymbol{\phi}^*||_2} = 0. \tag{2.54}$$

As the name suggests, this rate is faster than linear, but slower than quadratic convergence.

Step length selection

In the previous section we have outlined the main principles for selecting a search direction $\boldsymbol{p}_t$, but in order to solve (or rather approximate) Eq. (2.45), we still have to pick a step length η, also known as line search.

Line search algorithms typically work by trying out a number of candidate step sizes, and stopping when certain criteria are met. In this section, we briefly introduce the stopping criteria used by the BFGS-based solvers applied later in this thesis, as well as a simple line search algorithm. For more complicated (but efficient) line search

algorithms, we refer to e.g., [62] Section 3.4, where the line search procedure that is used in most BFGS-based solvers (L-BFGS included) is presented.

The most simple termination criterion for a step length η would be $f(\boldsymbol{\phi}_t + \eta \boldsymbol{p}_t) < f(\boldsymbol{\phi})$, i.e., that the function value decreases. However, this criterion alone is known to yield slow convergence, since it does not ensure that the function value is decreased enough or *sufficiently* each step. The condition of *sufficient decrease*, also known as the *Armijo condition*, states that

$$f(\boldsymbol{\phi}_t + \eta \boldsymbol{p}_t) \leq f(\boldsymbol{\phi}_t) + c_1 \eta \nabla f_t^T \boldsymbol{p}_t, \tag{2.55}$$

for some constant $c_1 \in (0, 1)$. Informally it states that not only should the function value be reduced, the reduction should be proportional to the directional derivative $\nabla f_t^T \boldsymbol{p}_t$. Termination based on Eq. (2.55) thus rules out some step sizes. However, it does not rule out very small step sizes, since $\boldsymbol{p}_t$ is a descent direction so Eq. (2.55) will be always satisfied for some small enough η, and leads to very slow progress. A second condition, called the *curvature condition*, is

$$\nabla f(\boldsymbol{\phi}_t + \eta \boldsymbol{p}_t)^T \boldsymbol{p}_t \geq c_2 \nabla f_t^T \boldsymbol{p}_t, \tag{2.56}$$

for some constant $c_2 \in (c_1, 1)$ with c_1 being the constant from Eq. (2.55). Informally, Eq. (2.56) states that the slope at the new point $f(\boldsymbol{\phi}_t + \eta \boldsymbol{p}_t)^T$ should be at least equal to c_2 times the current slope. Intuitively, this means that if the candidate point does not have enough negative slope, we do not pursue that direction further and terminate the line search; on the other hand, if the slope at the candidate point is strongly negative, we expect that we can minimize f further by exploring higher step sizes. Hence, combining the curvature condition with sufficient decrease, the line search does not terminate with really small step sizes, if there is the prospect of decreasing f further in the current search direction.

The two conditions Eq. (2.55) and Eq. (2.56) are collectively called the *Wolfe conditions* and are often the backbones of line search methods. Referring to [62], c_1 is often chosen as some very small value, e.g., $c_1 = 10^{-4}$, and for Newton or quasi-Newton methods, $c_2 = 0.9$ is typical.

From the curvature condition Eq. (2.56), we see that this requires the line search method to evaluate the gradient $\nabla f(\boldsymbol{\phi}_t + \eta \boldsymbol{p}_t)$ numerous times, which can get costly. An alternative strategy for avoiding unnecessarily small step sizes is to use a *backtracking* strategy and evaluate only the sufficient decrease condition Eq. (2.55). The idea of backtracking, is to start with some "non-small" step length $\bar{\eta}$ and evaluate Eq. (2.55); if the condition is satisfied, we terminate line search, and if not, we decrease $\bar{\eta}$ and try again. As long as the decrease in $\bar{\eta}$ converges towards zero in a finite number of evaluations, we are guaranteed that the line search also terminates in a finite number of evaluations. A typical backtracking algorithm is given below.

function BACKTRACKING LINE SEARCH($\beta, \gamma \in (0, 1)$)
  $n \leftarrow 0$
  repeat

$\eta \leftarrow \beta^n$
$n \leftarrow n + 1$
until $f(\boldsymbol{\phi}_t + \eta \boldsymbol{p}_t) \leq f(\boldsymbol{\phi}_t) + c_1 \eta \nabla f_t^T \boldsymbol{p}_t$ is satisfied.
return $\boldsymbol{\phi}_t + \eta \boldsymbol{p}_t$
end function

In order for Newton and quasi-Newton methods to achieve the rapid rates of convergence, that they are known for, the initial step size of 1 (as is implicitly the case in the above algorithm) should always be used ([62] Section 3.4). For quasi-Newton method, some extra care must be taken, when designing a line search algorithm, since the search direction can, in some cases, become a non-descent direction. A typical fix is to limit n to some maximum integer, thus sometimes possibly accepting a tiny step in a non-descent direction.

Regularization and differentiability

So far we have assumed that f is differentiable and how it then makes sense to approximate the curvature in order to speed up convergence. If we now return to the optimization of a regularized optimization problem Eq. (2.44), this objective function is differentiable if both $L(\cdot)$ and $\Omega(\cdot)$ are differentiable. In Section 2.2 we introduced ℓ^1 and ℓ^2 regularization as a means to control overfitting of the loss function. While ℓ^2 is differentiable, we see that the ℓ^1 function is not differentiable in zero. Consequently, Newton, BFGS, and L-BFGS are not suited for ℓ^1 regularized losses and in this section we briefly introduce a method that builds on L-BFGS, but which is able to handle the non-differentiable ℓ^1.

The orthant-wise limited-memory quasi-Newton (OWL-QN) is introduced in [4], which we refer to for the full details of the method. Here follows an informal introduction to its workings.

While ℓ^1 is not differentiable in zero, if we restrict the parameter vector to an *orthant*, i.e., the half-space where the parameters maintain their sign, it is differentiable and the gradient given the orthant is a linear function of its argument so its second derivatives are zero. Making this observation, OWL-QN then proceeds by noting that any second-order behavior given an orthant containing the current iterate comes from the loss function alone. Therefore, by constraining the search direction to a projection onto the orthant defined by the sign pattern of the current coordinates, the L-BFGS approximation to the inverse Hessian of the loss alone can be used. Additionally, only a slight modification to the line search procedure needs to be made, constraining it to not move the coordinates out of the current orthant, i.e., setting to zero any coordinates that change sign prior to evaluating the termination conditions. In [4] as well as the implementation of OWL-QN that we use, a backtracking line search similar to the one shown in the previous section is adapted to accommodate the constraint.

Geometry of loss functions

We sum up this section by briefly discussing some implications of the properties of the loss function $L(\cdot)$. If and when the loss function in Eq. (2.44) is a *convex* function, informally meaning that it has a unique minimum, but it may have multiple minimizers, then Eq. (2.44) with ℓ^1 and ℓ^2 is also convex, since both these regularizers are also both convex. Quasi-Newton based approaches are guaranteed convergence to a *local minimum*, assuming the function being minimized is also *Lipschitz continuous*, a property which loosely states that there exists a definite real number that bounds the absolute value of the slope of a line connecting any two points of the function. Thus, when the objective function is convex and Lipschitz continuous, quasi-Newton converges to a global minimum, since every local minimum of a convex function is also a global minimum.

Now, for non-convex loss functions, we assume these are Lipschitz continuous, so quasi-Newton converges to a local minimum. However, since the loss function is non-convex, we do not know whether a local minimum is also a global minimum. In practice, we either make do with a local minimum or we run multiple random restarts and empirically observe how different initial parameterizations affect the solution.

Stochastic gradient descent

The two techniques for large-scale optimization that we have introduced so far, L-BFGS and OWL-QN, are categorized as *batch learning* algorithm, named as such because they perform updates to the iterate each iteration based on the entire data set. A competing strategy is *stochastic gradient descent* (SGD) or *on-line* learning, where instead each update to the iterate is a stochastic approximation of the gradient. Stochastic gradient descent can be applied to optimization problems where the objective function $f(\boldsymbol{\phi})$ can be expressed as a sum,

$$f(\boldsymbol{\phi}) = \sum_{n=1}^{N} f_n(\boldsymbol{\phi}), \tag{2.57}$$

where n typically indexes the training examples and each function $f_n(\cdot)$ is associated with a single observation. The minimization problem Eq. (2.30) of a penalization term and a negative log-likelihood with iid. observations can be seen to be an instance of such a function. In the batch methods presented previously, minimization of the objective function is performed as steps along the negative gradient, which due to the formulation of f as a sum becomes

$$\boldsymbol{\phi}_{t+1} = \boldsymbol{\phi}_t - \eta_t \sum_n \nabla_\phi (f_n)_t, \tag{2.58}$$

where η_t is a step length that can vary in each iteration, such as in (quasi-)Newton methods. In stochastic gradient descent, one instead picks a *single data point* and

makes an update of the iterate using an approximate gradient *based on that data point alone*, i.e.,

$$\boldsymbol{\phi}_{t+1} = \boldsymbol{\phi}_t - \eta_t \nabla_{\phi}(f_n)_t. \tag{2.59}$$

Updates are then performed for each data point with multiple passes over data in random order, each called an *epoch*, until convergence. SGD can be made more efficient by performing updates based on *mini-batches*, i.e., instead of a gradient approximation based on one sample at the time, use more samples.

Being a gradient-based algorithm, the non-differentiable ℓ_1 penalty function must also be specially handled in SGD. One strategy very similar to OWL-QN is presented by Tsuruoka et al. [70] and more recently very fast SGD training algorithms with ℓ_1 penalization and adaptive learning rates have been presented [52, 53].

2.3 Bayesian inference

We now take a step back and once again regard Eq. (2.19) from Section 2.1, i.e., Bayes theorem

$$p(\boldsymbol{\phi}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{\phi})p(\boldsymbol{\phi})}{p(\mathcal{D})}. \tag{2.60}$$

In the approach we now focus on, called *generative models*, one specifies the likelihood $p(\mathcal{D}|\boldsymbol{\phi})$ and prior $p(\boldsymbol{\phi})$ as probability distributions. Generative models are named after the fact that by sampling from them we can simulate data in input space.

Ideally, we would like to *infer* exactly the distribution $p(\boldsymbol{\phi}|\mathcal{D})$ using training data, and then subsequently use the posterior to make optimal *decisions*. We refer to these two stages as the *inference stage* and the *decision stage*, respectively. In the following we focus on the inference stage.

Assuming that we cannot do exact inference of $p(\boldsymbol{\phi}|\mathcal{D})$, in the following we introduce the Markov Chain Monte Carlo (MCMC) sampling technique and how that can be used for doing approximate inference.

Markov Chain Monte Carlo

Markov Chain Monte Carlo is a sampling technique where samples $\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, \ldots$ form a *Markov chain*. Here we will just focus on discrete random variables, however everything in this section holds for continuous random variables as well, provided summation is turned in to integration. We first define a few things about Markov chains.

Markov chains

A (first order) Markov chain is a sequence of random variables $\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, \ldots$ where the conditional distributions $p(\boldsymbol{z}^{(m+1)}|\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)})$ for all $m \in \{1, \ldots, M-1\}$ meet the following conditional independence property

$$p_m(\boldsymbol{z}^{(m+1)}|\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}) = p_m(\boldsymbol{z}^{(m+1)}|\boldsymbol{z}^{(m)}). \tag{2.61}$$

We call a Markov chain *homogeneous* if it has stationary transition probabilities. That is $p_m(\boldsymbol{z}^{(m+1)}|\boldsymbol{z}^{(m)}) = p(\boldsymbol{z}^{(m+1)}|\boldsymbol{z}^{(m)})$ for all m. For our brief treatment of MCMC, we are only interested in homogeneous Markov chains.

The joint distribution of a homogeneous Markov chain $\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, \ldots$ is determined by

(i) The *initial distribution*, $p(\boldsymbol{z}^{(0)})$

(ii) The stationary transition probabilities, $p(\boldsymbol{z}^{(m+1)}|\boldsymbol{z}^{(m)})$.

The marginal distribution in state $\boldsymbol{z}^{(m+1)}$ is then given as

$$p(\boldsymbol{z}^{(m+1)}) = \sum_{\boldsymbol{z}^{(m)}} p(\boldsymbol{z}^{(m+1)}|\boldsymbol{z}^{(m)})p(\boldsymbol{z}^{(m)}). \tag{2.62}$$

When the transition probabilities of the Markov chain *preserve* the initial distribution, i.e.,

$$p^*(\boldsymbol{z}) = \sum_{\boldsymbol{z}'} p(\boldsymbol{z}|\boldsymbol{z}')p^*(\boldsymbol{z}'), \tag{2.63}$$

or in other words, the marginal distribution of a state $\boldsymbol{z}$ is invariant w.r.t. the transition probabilities, we call $p^*(\boldsymbol{z})$ *invariant* to the Markov chain. All Markov chains used in MCMC have an invariant distribution.

An important property of Markov chains for MCMC is if it satisfies *detailed balance*, which states

$$p(\boldsymbol{z}^{(m+1)}|\boldsymbol{z}^{(m)})p(\boldsymbol{z}^{(m)}) = p(\boldsymbol{z}^{(m)}|\boldsymbol{z}^{(m+1)})p(\boldsymbol{z}^{(m+1)}). \tag{2.64}$$

Detailed balance implies $p^*(\boldsymbol{z})$ is invariant since

$$\sum_{\boldsymbol{z}'} p(\boldsymbol{z}|\boldsymbol{z}')p^*(\boldsymbol{z}') = \sum_{\boldsymbol{z}'} p(\boldsymbol{z}'|\boldsymbol{z})p^*(\boldsymbol{z}) = p^*(\boldsymbol{z}) \sum_{\boldsymbol{z}'} p(\boldsymbol{z}'|\boldsymbol{z}) = p^*(\boldsymbol{z}). \tag{2.65}$$

Detailed balance is therefore a sufficient (but not necessary) condition to ensure $p^*(\boldsymbol{z})$ is invariant to the Markov chain. Thus, if we can prove for some Markov chain that detailed balance is satisfied, the initial distribution is invariant.

In MCMC we want the distribution we would like to sample from, the desired distribution, to be invariant. Here we can use detailed balance as a sufficient condition. With an additional requirement that for $m \to \infty$, the marginal $p(\boldsymbol{z}^{(m)})$ converges to the invariant distribution, regardless of the choice of initial distribution $p(\boldsymbol{z}^{(0)})$ - a property of Markov chains called *ergodicity* - then in the limit $m \to \infty$ the chain generates samples from the desired distribution. For an ergotic Markov chain, the invariant distribution is also known as the *equilibrium distribution*.

The Metropolis-Hastings algorithm

The original *Metropolis* algorithm was introduced in 1953 by [56], but later generalized as the *Metropolis-Hastings* algorithm in 1970 by [38]. Similar to other sampling techniques (e.g., rejection and importance sampling, [11] Section 11.1), in Metropolis-Hastings samples are drawn from a proposal distribution, q, which it is easy to obtain samples from. However, in Metropolis-Hastings the proposal distribution $q_k(\boldsymbol{z}|\boldsymbol{z}^{(\tau)})$ is conditionally dependent on the current state $\boldsymbol{z}^{(\tau)}$. Also, in Metropolis-Hastings q is not assumed symmetric, hence we show the dependence on the proposal distribution with the subscript k. At time τ, a candidate sample $\boldsymbol{z}^*$ is drawn from $q_k(\boldsymbol{z}|\boldsymbol{z}^{(\tau)})$ and this is accepted with acceptance probability

$$A_k(\boldsymbol{z}^*, \boldsymbol{z}^{(\tau)}) = \min\left(1, \frac{\tilde{p}(\boldsymbol{z}^*)q_k(\boldsymbol{z}^{(\tau)}|\boldsymbol{z}^*)}{\tilde{p}(\boldsymbol{z}^{(\tau)})q_k(\boldsymbol{z}^*|\boldsymbol{z}^{(\tau)})}\right), \tag{2.66}$$

where $\tilde{p}$ is proportional to the desired distribution p by the following relation

$$p(\boldsymbol{z}) = \frac{\tilde{p}(\boldsymbol{z})}{Z_p}, \tag{2.67}$$

i.e., we assume that we can easily evaluate $p(\boldsymbol{z})$ up to a normalization constant Z_p via $\tilde{p}(\boldsymbol{z})$. If the proposal distribution is symmetric, such that $q(\boldsymbol{z}|\boldsymbol{z}^{(\tau)}) = q(\boldsymbol{z}^{(\tau)}|\boldsymbol{z})$ for all $\boldsymbol{z}$, then Eq. (2.66) reduces to the original Metropolis algorithm.

The fact that the desired distribution $p(\boldsymbol{z})$ is an invariant distribution to the Markov chain generated by Metropolis-Hastings, can be seen by showing that detailed balance is satisfied. First note that in the terminology of Markov chains the stationary transition probability distribution (for a specific k) for Metropolis-Hastings is

$$p_k(\boldsymbol{z}'|\boldsymbol{z}) = q_k(\boldsymbol{z}'|\boldsymbol{z})A_k(\boldsymbol{z}, \boldsymbol{z}'). \tag{2.68}$$

We insert the above into the right-hand side of Eq. (2.64) and then use Eq. (2.66) to get,

$$\begin{aligned} p(\boldsymbol{z})q_k(\boldsymbol{z}'|\boldsymbol{z})A_k(\boldsymbol{z}, \boldsymbol{z}') &= \min(p(\boldsymbol{z})q_k(\boldsymbol{z}'|\boldsymbol{z}), p(\boldsymbol{z}')q_k(\boldsymbol{z}|\boldsymbol{z}')) && (2.69)\\ &= \min(p(\boldsymbol{z}')q_k(\boldsymbol{z}|\boldsymbol{z}'), p(\boldsymbol{z})q_k(\boldsymbol{z}'|\boldsymbol{z})) && (2.70)\\ &= p(\boldsymbol{z}')q_k(\boldsymbol{z}|\boldsymbol{z}')A_k(\boldsymbol{z}', \boldsymbol{z}), && (2.71) \end{aligned}$$

thus proving detailed balance.

In order for the invariant distribution to be the equilibrium distribution, as is required for the MCMC sampler to work, there are weak restrictions on $q_k(\boldsymbol{z}'|\boldsymbol{z})$ and $p(\boldsymbol{z})$ that ensure the Markov chain generated by Metropolis-Hastings is ergotic [60].

Gibbs sampling

Gibbs sampling was introduced in 1984 by [30] without any connection to the Metropolis-Hastings algorithm. As we shall soon show, it is however a specialization of Metropolis-Hastings.

In Gibbs sampling we sample $p(\boldsymbol{z}) = p(z_1, \ldots, z_M)$ by sampling one variable (or a block) at the time, while conditioning on the remaining. Formally, we assume that we have an initial state $\boldsymbol{z}$ for the Markov chain. Then in each step, we pick z_i and replace that with a sample drawn from $p(z_i|\boldsymbol{z}_{\setminus i})$, where $\boldsymbol{z}_{\setminus i}$ denotes all the variables $z_j, j \neq i$. The algorithm then proceeds either by cycling through the variables or by choosing a variable each step at random from some distribution.

To see that Gibbs sampling is a valid MCMC sampler, we first note that detailed balance holds by consequence of the product rule

$$p(z_i|\boldsymbol{z}_{\setminus i})p(\boldsymbol{z}_{\setminus i}) = p(z_i, \boldsymbol{z}_{\setminus i}) = p(\boldsymbol{z}_{\setminus i}|z_i)p(z_i). \tag{2.72}$$

Ergodicity of the Markov chain also needs to be proven, which ensures the chain converges to the desired distribution $p(\boldsymbol{z})$, regardless of initialization. A sufficient

condition for ergodicity is that the conditional distributions are never near zero. When that is not the case, ergodicity should be proven. Alternatively, random restarts and empirical assessment of the resulting samples are frequently used to evaluate a Gibbs sampler where ergodicity is not proven.

We can frame Gibbs sampling as a special case of Metropolis-Hastings. First assume that the current state is $p(\boldsymbol{z}_{\setminus i})$ and we are sampling a proposal $\boldsymbol{z}^*$ from $p(z_i^*|\boldsymbol{z}_{\setminus i})$. We note that $p(\boldsymbol{z}_{\setminus i})$ remains fixed after the step, i.e., $\boldsymbol{z}^*_{\setminus i} = \boldsymbol{z}_{\setminus i}$. Also, $p(\boldsymbol{z}) = p(z_i|\boldsymbol{z}_{\setminus i})p(\boldsymbol{z}_{\setminus i})$, so the step is accepted with probability

$$A_i(z_i, \boldsymbol{z}_{\setminus i}) = \min\left(1, \frac{p(\boldsymbol{z}^*)q_i(\boldsymbol{z}|\boldsymbol{z}^*)}{p(\boldsymbol{z})q_i(\boldsymbol{z}^*|\boldsymbol{z})}\right) = \min\left(1, \frac{p(z_i^*|\boldsymbol{z}^*_{\setminus i})p(\boldsymbol{z}^*_{\setminus i})p(z_i|\boldsymbol{z}^*_{\setminus i})}{p(z_i|\boldsymbol{z}_{\setminus i})p(\boldsymbol{z}_{\setminus i})p(z_i^*|\boldsymbol{z}_{\setminus i})}\right) = 1.$$

Hence, the Gibbs step is always accepted.

Inference using Gibbs sampling

In this thesis we will be using Gibbs sampling to draw samples from the posterior, thus inferring parameters from data, in a couple of ways.

The first method we introduce relies on *conjugate priors*. For any distribution in the *exponential family*, these all have a *conjugate prior*. When a model is specified with a likelihood $p(\mathcal{D}|\boldsymbol{\phi})$ from the exponential family and a corresponding conjugate prior as $p(\boldsymbol{\phi})$, i.e., $\boldsymbol{\phi}$ are continuous random variables, then the posterior distribution $p(\boldsymbol{\phi}|\mathcal{D})$ can be specified analytically and will take the same form as the prior. Hence, we can apply our Gibbs sampling steps using this posterior, as long as we can efficiently sample from it. For more information on exponential distributions and conjugacy, we refer to [11] Section 2.4.

When the parameters we are interested in are discrete random variables, as the z_i's used in the preceding sections, the fact that the normalization constant in Bayes theorem is a sum is important. Let us rephrase Bayes theorem for a single discrete random variable z_i,

$$p(z_i|\boldsymbol{z}_{\setminus i}) = \frac{p(\boldsymbol{z}_{\setminus i}|z_i)p(z_i)}{\sum_{i'} p(\boldsymbol{z}_{\setminus i'}|z_{i'})p(z_{i'})} = \frac{p(z_i, \boldsymbol{z}_{\setminus i})}{\sum_{i'} p(z_{i'}, \boldsymbol{z}_{\setminus i'})}. \tag{2.73}$$

This enables an approach where we evaluate $p(z_i, \boldsymbol{z}_{\setminus i})$ for each i, then compute the sum in the denominator explicitly, thus obtaining a posterior sample. For many parameters, continually evaluating the normalization constant after each step would be infeasible and instead we can do *blocked* Gibbs instead. In blocked Gibbs, the only modification to the basic algorithm is to update multiple parameters (i.e., a block) at the same time, which is more computationally efficient and does not alter the correctness of the sampler. Hence, for the above posterior evaluation, we can instead calculate the denominator once, while storing all the summands. Then once we have the normalization constant, we evaluate the posterior for each of the parameters. Blocked Gibbs sampling applies equally well for posterior updates based on conjugate priors.

Making decisions using Gibbs sampling

We have introduced Gibbs sampling as a means to infer parameters and at the same time we are sampling the posterior distribution, which suggests that we can use these samples to predict. Recall the predictive distribution from Eq. (2.25); in most cases, we cannot evaluate this analytically due to the integral and we need to approximate instead using a finite sample

$$p(y|\boldsymbol{x}^*, \mathcal{D}) \simeq \frac{1}{L} \sum_{l=1}^{L} p(y|\boldsymbol{x}^*, \boldsymbol{\phi}^{(l)}), \tag{2.74}$$

provided the samples $\boldsymbol{\phi}^{(l)}$ are drawn independently from $p(\boldsymbol{\phi}|\mathcal{D})$. In Gibbs sampling with one variable at a time, there is naturally a lot of correlation between samples, so successive samples from the Gibbs sampler are not suitable for estimating a finite sample expectation. On the other hand if we are able to sample directly from the joint distribution, we would get successive samples that are independent. Blocked Gibbs sampling, that we introduced earlier, is thus a middle-ground, where we hope to remove at least some of the independence between successive samples. Regardless of which type of Gibbs sampler we use, unless we are able to sample the joint distribution directly, the essence is not to estimate the finite sample expectation from successive samples. Instead, if we assume the sampler has converged to the equilibrium distribution, we can subsample a sequence drawn from the Gibbs sampler with equidistant samples and as long as the distance between subsamples is big enough, these samples will be independent enough for most practical purposes. In practice, in order for the chain to have converged before we start sub-sampling, we let the chain run initially for some number of iterations, which we call *burn-in*.

2.4 Evaluating click-through rate performance

We end this chapter with a small section on how we report the results in this thesis. As mentioned in the introduction (Chapter 1), the benchmarking application in this work is click-through rate prediction. Given a transaction log of interactions where each observation can be represented as a feature vector $\boldsymbol{x}$ and a binary label $\boldsymbol{y}$ representing a *view* (0) or a *click* (1), we assume a model which learns probabilities $p(Y = 1|\boldsymbol{x})$. Splitting the data into separate *train* and a *test* sets, our goal is to maximize the performance on the test set(s). The more independent test sets we can report performance on, the more we can say about generalization error and the more certain we can make our conclusions.

Adform's transaction logs are inherently sequential, which we would like to take into account when testing models. Therefore our strategy for measuring performance on held out data is the following: (i) Prepare a dataset of D days, (ii) train on the first $D - 1$ days, and (iii) report performance on the final day. This we can then repeat over a larger window with different test days, which then, assuming the test sets are independent, gives us an estimate of the generalization error.

The measures which we use on held out data to measure performance are *logistic loss* (LL) and *area under the curve* (AUC) of the receiver operating characteristic, which we briefly introduce in the following.

Logistic loss is defined as

$$-\sum_{n=1}^{N} y_n \log(p(Y_n = 1|\boldsymbol{x}_n)) + (1 - y_n) \log(1 - p(Y_n = 1|\boldsymbol{x}_n)), \tag{2.75}$$

where n indexes the samples of the test dataset. For reporting our results based on logistic loss, we use a variant normalized with respect to the average click-through rate on the training data, which we dub CTR_0:

$$LL := \frac{\sum_{n=1}^{N} y_n \log(p(Y_n = 1|\boldsymbol{x}_n)) + (1 - y_n) \log(1 - p(Y_n = 1|\boldsymbol{x}_n))}{\sum_{n=1}^{N} y_n \log(CTR_0) + (1 - y_n) \log(1 - CTR_0)}, \tag{2.76}$$

i.e., a number lower bounded by 0 and which should be lower than 1 in order for $p(Y_n = 1|\boldsymbol{x}_n)$ to be a better predictor than predictions using a random probability of CTR_0.

The area under the curve (AUC) of the receiver operating characteristic can be thought of as the accuracy of discriminative performance [6], and is a measure which takes into account *only* the relative order of observations under some model, not the particular values predicted by that model. The AUC however has the attractive property, that it is insensitive to class skew, i.e., changes in the number of positive versus negative examples [25].

When used for model comparison, LL and AUC thus complement each other. While a difference in LL can be merely a trivial problem with mis-calibrated probabilities, the AUC will capture which model outputs a better *relative* ordering between

observations. With an improvement in *both* measures, we will generally consider that model better, whereas when an improvement is detected only in AUC, this may indicate a problem of poorly calibrated model probabilities.

As we detailed in the introduction (Chapter 1), bids are calculated proportionally to the click-through rate prediction and are also governed by a throttling mechanism. In Adform we generally assume this throttling mechanism is able to rectify (at least slightly) mis-calibrated probabilities and thus lean towards the AUC measure as our preferred performance metric for model comparison. However, we also monitor logistic loss and prefer that both metrics are improved or at least the LL is not far off target.

There exist many other measures for classification, for instance based on confusion matrices, where *precision-recall* curves and *F-measures* are popular choices. There exists relationships between receiver operator characteristic and precision-recall curves and it has been argued, that precision-recall curves give a more informative picture of an algorithms performance when the target classes are highly skewed [17]. While in particular precision-recall curves would be fitting for our purposes, they require that we binarize the probabilities of a model, i.e., we need to define a threshold that converts the model predictions to classifications instead. This would present us with the additional choice of where to cut or we need to weigh or average multiple curves from different thresholds, i.e., something we are not interested in.

Training feedback loops

The fact that a click-through rate prediction model is involved in the data gathering process, i.e., we bid in order to win impressions and we only observe the true label when we win the auction, presents us with the possible challenge of "training feedback loops" [13]; or as it is also known in reinforcement learning, an "exploration vs. exploitation dilemma" [7, 66]. We argue however in the following that such feedback loops do not pose an issue for the experimental setup used in this thesis, and thus we do not touch upon for instance the multi-armed and contextual bandit settings [3, 9, 14, 31, 47, 66, 73], which seek to remedy this.

In Adform many real-time bidding setups are bidding *fixed prices* scattered in all price ranges (e.g., low, medium, and high), i.e., where *no model* is involved in the bidding process. This data contributes $\geq 80\%$ of the observations that models are trained and tested on and which we consider as being sufficiently "random" in nature, so that exploration is achieved. When trying out new features or methods, we have also not seen any evidence that feedback loops cause any issues with the production model being favored in terms of performance. These circumstances leads to the delimitation in this thesis, that we *do not* consider training feedback effects to present any issues with (lack of) exploration or performance bias, and thus we focus entirely on the aforementioned AUC and LL measured on held-out real-time bidding data.

CHAPTER 3

Models for click-through rate prediction

As we wrote in the introduction, the main area of application in this thesis is for click-through rate prediction. We remind briefly about our motivation (see also Chapter 1): Adform is bidding in real-time auctions on behalf of their customers (advertisers and agencies) using performance-based pricing models. In the case of the *cost-per-click* (CPC) valuation, bids are according to

$$\mathcal{B}(r) \propto CPC_t \cdot p(\text{click}|f(r)), \tag{3.1}$$

where the customer specifies CPC_t and $f(r) \mapsto \boldsymbol{x}$ is here a convenience function which based on the information in the bid request r outputs a feature vector, $\boldsymbol{x}$. By analogy, any improvement in the model we use for estimating the probability of click, $p(\text{click}|f(r))$, should translate to more optimal bidding.

The focus in this chapter is on training models for click-through rate prediction and has an emphasis on the kind of model used in Adform as well as our findings with a model extending the existing with *latent features*. While the working example in this chapter is click-through rate prediction, the techniques apply in many other settings and are not restricted to computational advertising in particular. Consequently, in Adform we apply the same modeling techniques for a number of other tasks as well, but which we deem out-of-scope for this thesis.

This chapter supplements the contribution Fruergaard [26, Appendix C] which is a part of this thesis.

3.1 Overview

The task that we are addressing in this chapter is the *supervised learning* problem of predicting the probability $p(y = 1|\boldsymbol{x}^*, \mathcal{D})$, where $y = \{0, 1\}$ represents a click (1) or not (0), $\boldsymbol{x}^*$ is a *feature vector* of attributes for the current observation and $\mathcal{D}$ is the historical data we have available, which consists of pairs $(\boldsymbol{x}_n, y_n), n = 1, \ldots, N$, i.e., previous feature vectors and corresponding labels.

Before we turn to describing the model applied in Adform for supervised learning, we here summarize some of the properties and assumptions about Adform's data and setup, which we argue narrows down the options of different modeling techniques to only a few.

Data and system

i All features are (or can be converted into) categorical variables.

- Continuous features such as for instance time can be discretized into, e.g., hour-of-day, day-of-week, workday/weekend, seasons, etc., and thus are converted to categorical variables.
- Lists of attributes, e.g., domains visited by a CookieId, can be cast as several categorical (binary) variables of the form (CookieId-has-attribute-1, CookieId-has-attribute-2, . . . , CookieId-has-attribute-K).

ii Different campaigns will use separate models.

iii Classes will be highly *skewed*, e.g., observations with a click will be much fewer than observations without a click.

iv Adform logs on the order of 100-thousands of different attributes. Far from all of them will be relevant for each campaign.

- Hence, the *sparser* the models, the less storage and I/O is required.

v Predictions should be *fast*, i.e., the model will be queried thousands of times per second.

vi Interpretability of the models is a plus, but not always a must.

vii Faster training times enable quicker adjustments to changes in market and/or segment dynamics, hence is a plus.

With these observations and constraints in mind, *sparse logistic regression* has become the model of choice for click-through rate prediction and other classification tasks in Adform. Logistic regression, in its most basic form, is a textbook method for binary classification, see for instance [11, p. 205]. Yet, with the above constraints in mind as well as logistic regression being a component in an extended model we describe later in this chapter, in the following we describe the model for sparse logistic regression that our experiments build on.

3.2 Sparse logistic regression

We can describe logistic regression as a probabilistic model of binary responses $\boldsymbol{y} = (y_1, \ldots, y_N)^T$ given the $P \times N$ *feature matrix* $\boldsymbol{X} = (\boldsymbol{x}_i, \ldots, x_N)$ with a Bernoulli distribution,

$$Y_n | \boldsymbol{x}_n \sim \text{Bern}(q_n) \tag{3.2}$$

or equivalently,

$$p(Y_n = y_n | \boldsymbol{x}_n) = q_n^{y_n} (1 - q_n)^{1 - y_n}, \tag{3.3}$$

and we choose $q_n = \sigma(\boldsymbol{\omega}^T \boldsymbol{x}) = 1/(1 - \exp(-\boldsymbol{\omega}^T \boldsymbol{x}))$, which is known as the *sigmoid* or *logistic function*. I.e., the probability of observing y_n is modeled as a non-linear function over a linear combination of the input features $\boldsymbol{x}_n$ with coefficients $\boldsymbol{\omega} = (\omega_1, \ldots, \omega_P)^T$. Assuming iid. observations, the complete-data likelihood thus is $\prod_n p(y_n|\boldsymbol{x}_n, \boldsymbol{\omega})$. In Section 2.2 we introduced *maximum posteriori* (MAP) optimization for machine learning problems and in particular the *sparsity inducing* ℓ^1 regularizer with the corresponding optimization problem Eq. (2.42). Now taking the log of the likelihood we get,

$$\log p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\omega}) = \sum_{n=1}^{N} y_n \log(q_n) + (1 - y_n) \log(1 - q_n). \tag{3.4}$$

The regularized optimization problem is then the following,

$$\underset{\boldsymbol{\omega}}{\operatorname{argmin}} \quad \lambda ||\boldsymbol{\omega}||_1 - \sum_{n=1}^{N} y_n \log(q_n) + (1 - y_n) \log(1 - q_n). \tag{3.5}$$

Since this optimization problem does not have a closed form solution, we resort to a sequential algorithm. Referring back to Section 2.2 we first note that the negative log-likelihood Eq. (3.4) is differentiable but the regularizer is not. This we discussed in Section 2.2 where we introduced the *quasi-Newton* algorithm OWL-QN that can be applied to this problem and requires only the gradient of the negative log-likelihood as well as the regularization strength λ. The required gradient is derived in Appendix A. For the regularization strength, λ, we pick a suitable value based on cross-validation where the held-out test sets are independent in time as described in Section 2.4.

It is often the case that the size of the target distributions, i.e., number of observations with $y_n = 1$ versus those with $y_n = 0$, is highly skewed. In these circumstances we can add an extra term to the coefficients vector and extent the data matrix X by an extra row taking always the value 1. This coefficient then acts as an *intercept* or *bias* weight and adjusts for the skewed targets. Since in CTR data, the clicks are indeed very rare compared to the number of non-clicks, we train logistic regression models using an intercept.

Predictions

Having run logistic regression on a training data set, we get the MAP coefficients $\boldsymbol{\omega}_{MAP}$, which for notational convenience we just refer to as $\boldsymbol{\omega}$ in the following. Since the expected value of a Bernoulli distributed random variable Y is $\mathbb{E}[Y] = q$, where $q \in (0, 1)$ is the Bernoulli parameter, we simply compute the prediction of a new observation with feature vector $\boldsymbol{x}^*$ as

$$\mathbb{E}[Y] = \sigma(\boldsymbol{\omega}^T \boldsymbol{x}^*) = \frac{1}{1 + \exp(-\sum_p \omega_p x_p)}. \tag{3.6}$$

I.e., the prediction is a point estimate reflecting that logistic regression is a *discriminative method*, meaning that we do not sample the posterior distribution but rather predict using the mode of the distribution, $\boldsymbol{\omega}_{MAP}$.

Computing predictions becomes particularly efficient when the feature vectors are sparse and binary, as well as the coefficient vector $\boldsymbol{\omega}$ being sparse. In this case the denominator in Eq. (3.6) only involves summing up the non-zero coefficients at the indices where $\boldsymbol{x}^*$ is nonzero, i.e., multiplication is not involved, making the logistic function particularly useful for real-time system such as RTB.

A further observation can make real-time predictions even more efficient: Consider the following reformulation of Eq. (3.6),

$$\frac{1}{1+\exp(-\sum_p \omega_p x_p)} = \frac{1}{1+\prod_{p' \in nnz(\boldsymbol{\omega}) \bigcap nnz(\boldsymbol{x}^*)} \exp(\omega_{p'})}, \tag{3.7}$$

where we use $nnz(\cdot)$ to denote the set of non-zero elements of the argument. This formulation opens up to the possibility of storing the exponentiated coefficients in a database, retrieving them in real-time and performing the product in real-time. This may not in itself be an optimization, however, in RTB, for instance, there is always a CookieId as part of a bid request and this CookieId (if known by Adform) maps to a subset of the features in $\boldsymbol{\omega}$. With the above formulation, that means that for the part of the feature vector that is cookie-related, the product can be precomputed, such that a single coefficient is stored per cookie in the database. The same can be done for other parts of the coefficient vector that relate to other fixed entities of a bid-request. In that way, predictions can be computed in real-time involving only a few multiplications thus becoming more efficient than computations based on Eq. (3.6), however at the cost of more storage in the database. This optimization only remains efficient, though, as long as the model coefficients $\boldsymbol{\omega}$ are not changed too frequently, otherwise, e.g., enumerating all CookieIds quickly becomes inhibiting.

3.3 Latent feature log-linear model

The sparse logistic regression model introduced in the previous section is a function over explicitly known features of the observations to a probability of an action, e.g., click. In this section we motivate an extension to the explicit features by introducing a matrix factorization model over latent features, thereby augmenting the feature space.

The *latent feature log-linear model* (LFL) by Menon and Elkan [54] is introduced as a model for *dyadic prediction*, which is the task of predicting an outcome (or label) for a *dyad*, (i, j), where entities of the data are uniquely identified by $i = 1, \ldots, I$ and $j = 1, \ldots, J$. If we focus on the particular case that labels of a training set $\boldsymbol{y} = y_1, \ldots, y_N$ are binary and we wish to predict the probability $p(y^* = 1|(i, j))$ for

a new observation, then the maximum likelihood estimator $p_{ML}(i,j)$ would be,

$$p_{ML}(y^*|(i^*,j^*)) = \frac{\sum_{n=1}^{N} y_n \delta_{i_n,i^*}\delta_{j_n,j^*}}{\sum_{n=1}^{N} \delta_{i_n,i^*}\delta_{j_n,j^*}} = \frac{C_{ij}}{V_{ij}}, \tag{3.8}$$

where we have introduced the matrices $\boldsymbol{C}$ with elements C_{ij} and $\boldsymbol{V}$ with elements V_{ij}. I.e., we just count the number of occurrences in data with (i^*,j^*) to get the denominator and in the nominator we count only the occurrences with $y_n = 1$. The maximum likelihood estimator defined above has the obvious problem of being undefined when V_{ij}. Besides, when $C_{ij} = 0, V_{ij} > 0$, p_{ML} predicts strictly zero although it might very well be that this probability is non-zero, but that not enough events with (i^*,j^*) for the positive case to have been observed.

This motivates the use of a *collaborative filtering* (CF) model where predictions involving the dyad (i^*,j^*) combine information also from (i^*,j) for some or all j and (i,j^*) for all or some i. A classic CF problem is the Netflix rating problem [8] where predicting ratings for an unseen (user,movie) pair can factor in the known ratings from other users that have seen similar movies.

Latent matrix factorization

As mentioned above the maximum likelihood estimator is ill suited for dyads without or with only few observations. Instead suppose there exists latent features $\boldsymbol{\alpha}_i \in \mathbb{R}^{(K)}$ for each i and $\boldsymbol{\beta}_j \in \mathbb{R}^{(K)}$ for each j and we introduce a *matrix factorization* model $p_{ij}^{MF} := p(y = 1|\boldsymbol{\alpha}_i, \boldsymbol{\beta}_j) = \sigma(\boldsymbol{\alpha}_i^T \boldsymbol{\beta}_j)$, i.e., the logistic function mapping the continuous values of $\boldsymbol{\alpha}_i^T \boldsymbol{\beta}_j$ into the interval $(0,1)$. From Eq. (3.4) we can then write the log-likelihood of a Bernoulli distribution over the targets y,

$$\sum_{n=1}^{N} y_n \log(p_{i_n j_n}^{MF}) + (1 - y_n) \log(1 - p_{i_n j_n}^{MF}). \tag{3.9}$$

Noting how the indices i_n and j_n index into I and J respectively, and with the count matrices $\boldsymbol{C}$ and $\boldsymbol{V}$ introduced as in the previous section, we can reformulate the above,

$$\sum_{(i,j)\in\mathcal{O}} C_{ij} \log(p_{ij}^{MF}) + (V_{ij} - C_{ij}) \log(1 - p_{ij}^{MF}), \tag{3.10}$$

where we have introduced the set $\mathcal{O}$ of every distinct pair (i,j) that occur in a dataset. This is an identical formulation of what Menon et al. [55] dubs a *confidence-weighted factorization*. This reformulation can be a significant performance optimization, since the number of distinct dyads is often much smaller than the total number of observations.

Introducing the matrices $\boldsymbol{A} = (\boldsymbol{\alpha}_1^T, \ldots, \boldsymbol{\alpha}_I^T)$ and $\boldsymbol{B} = (\boldsymbol{\beta}_1^T, \ldots, \boldsymbol{\beta}_I^T)$, we continue in the same manner as logistic regression and formulate an optimization problem

using Eq. (3.10),

$$\underset{\boldsymbol{A},\boldsymbol{B}}{\operatorname{argmin}} \quad \Omega(\boldsymbol{A},\boldsymbol{B}) - \Big(\sum_{(i,j)\in\mathcal{O}} C_{ij}\log(p_{ij}^{MF}) + (V_{ij}-C_{ij})\log(1-p_{ij}^{MF})\Big), \tag{3.11}$$

with $\Omega(\boldsymbol{A},\boldsymbol{B})$ being some regularization function over the latent factors. This optimization problem is no longer convex, i.e., it may have many local minima. Regularization helps towards excluding some minima, but in general one has to resort to for instance random restarts and empirically assess the different solutions. The optimization problem is however convex in $\boldsymbol{A}$ for fixed $\boldsymbol{B}$, and vice versa, which can be exploited in an implementation.

Incorporating side-information

In the scenarios where we are interested in dyadic predictions based on the collaborative filtering model introduced in Section 3.3, for each dyadic observation (i, j) there are other attributes (side-information) available as well, which we would like to use also in order to improve the predictive performance. In our contribution [26, Section 2.3] (Appendix C) we show how this can be done using a logistic regression model identical to the one introduced in Section 3.2. For brevity here we only note, that combining both the LFL model with a side-information model involves alternating between training the latent features while keeping the side-information model fixed, and vice versa.

Computation

For training the LFL model, we can use the numerical optimization techniques introduced in Section 2.2. The choice of solver depends on the type of regularization function $\Omega(\boldsymbol{A},\boldsymbol{B})$ we choose. Following the original paper [54], we can add ℓ_2 regularization to the latent factors as,

$$\Omega_{\ell_2}(\boldsymbol{A},\boldsymbol{B}) = \lambda_\alpha \sum_{i=1}^{I} ||\boldsymbol{\alpha}_i||_2^2 + \lambda_\beta \sum_{j=1}^{J} ||\boldsymbol{\beta}_j||_2^2, \tag{3.12}$$

or we can gather both penalization weights in the above under one $\lambda_{\alpha\beta} := \lambda_\alpha = \lambda_\beta$. For ℓ_2-based regularization, the objective function remains differentiable, hence we can use an L-BFGS quasi-newton solver.

If we instead introduce ℓ_1 based regularization, thereby favoring sparse latent factors, i.e.,

$$\Omega_{\ell_1}(\boldsymbol{A},\boldsymbol{B}) = \lambda_\alpha \sum_{i=1}^{I} ||\boldsymbol{\alpha}_i||_1 + \lambda_\beta \sum_{j=1}^{J} ||\boldsymbol{\beta}_j||_1, \tag{3.13}$$

or using a single penalization weight $\lambda_{\alpha\beta} := \lambda_\alpha = \lambda_\beta$ and we may use the OWL-QN quasi-newton solver instead.

For really large problems it is possible instead to use on-line learning based on stochastic gradient descent, as we also discussed in Section 2.2.

3.4 LFL versus other CF techniques

In a standard recommender system, a database holds information of users and their preferences for various items, and the systems' responsibility is to suggest new items to users that they will probably like. We already mentioned Netflix [8] as a typical example of such a system. Collaborative filtering (CF) is a successful approach to this problem, which only uses the preferences in the database to make recommendations based on "similarity" of users and items, where "similarity" refers to the *latent* aspects which are learned by the model.

In Eq. (3.8) we phrased click-through rate prediction as a dyadic prediction task where we can see p_{ML} over all pairs (i, j) as a matrix, say $\boldsymbol{X}$, where most entries X_{ij} are missing. I.e., the problem can be seen as a collaborative filtering setup which we can approach using matrix factorization, i.e., by introducing latent k-dimensional features $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ to model the observed entries as $X_{ij} \approx \boldsymbol{\alpha}_i^T \boldsymbol{\beta}_j$. This however has a couple of shortcomings with respect to our problem formulation: (i) The outputs $\boldsymbol{\alpha}_i^T \boldsymbol{\beta}_j$ may not be proper probabilities and (ii) the factorization does not take into account the confidence in the observed entries, i.e., observations with few views are weighted equally to those with many. These are two main points, which are addressed by the model from [55] and which we have introduced in the preceding sections.

The third very important inclusion, is that of *side-information*. Whereas in CF, side-information is traditionally seen as being a *secondary* source of information to the *primary* source from expressed preferences, in the case of click-through rate datasets such as we analyze, there is generally an abundance of explicit features (side-information in CF) which can be coupled to targets through a supervised learning framework, such as the one introduced in Section 3.2. That leads to the opposite scenario, where it is natural to *first* learn as much as possible from explicit features, and only *secondarily* try to exploit recommender effects. Therefore, explicit features should be an integral part of the model, as they are in the combined LFL framework that we introduced in Section 3.3.

3.5 Results on click-through rate data

In our contribution [26] (Appendix C) we review the combined model from the logistic regression model presented in Section 3.2 for learning coefficients for the side-information and the latent factor model as introduced in Section 3.3. We henceforth refer to this model as LR+LFL$_K$, where K refers to the latent factor dimensionality and $K = 0$ is a special case.

Our purpose is to investigate the performance effects of adding latent features through the LFL framework and comparing that to a model based on side-information

alone, where the latter is just the sparse logistic regression model explained in Section 3.2.

The entities that we consider in the LFL model are pairs of *domains* of the web sites and *banner ads*, i.e., we can think of this as a collaborative filtering setup where the *users* are *domains* and *items* are *banners*.

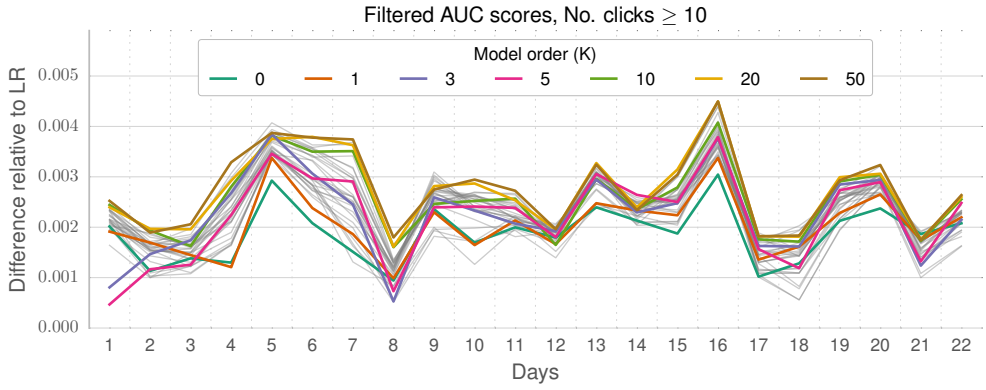
In Fig. 3.1 we show the most important results from [26]. The topmost Fig. 3.1(a) summarizes on a daily basis the *difference* between the average AUC score per ad for a number of instantiations of the LR+LFL$_K$ model with varying model order K and the performance of the side-information model alone, i.e., the zero-axis is set from the side-information model as the baseline and a positive difference signifies an improvement. It can be seen based on this plot, that all model orders, including the special case $K = 0$, improve on the baseline and also that model orders $K = 20, 50$ are leading *almost* all of the time.

In the figures Fig. 3.1(b) and Fig. 3.1(c), we further summarize the results for all days in box plots and measuring performance differences in AUC (b) and logistic loss (c), respectively.

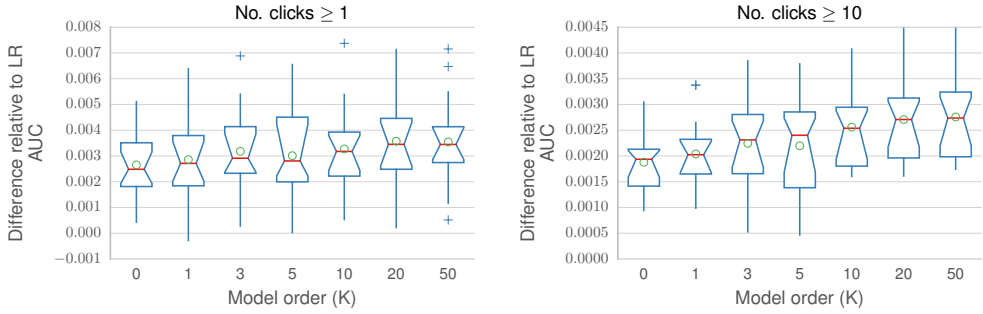
As we note in the work, as we are measuring performances on data from each banner ad separately, then averaging over them, it plays a role on the average if we filter out some banner ads. In the case of AUC scores, this measure is not defined without observations of the positive class (i.e., clicks), hence if we include only the banner ads with clicks each day, we get the left plot in Fig. 3.1(b). If we filter *more* for inclusion each day in terms of minimum number of clicks, the AUC scores become less noisy since they are computed over more observations, and the variance of the averages decrease. Hence, in the right most plot of Fig. 3.1(b), the box plot is shown with per day averages only including those banner ads with 10 or more clicks. In the latter case, the number of banner ad (i.e., independent test sets) performances included in the per day averages is still 400-500.

For the logistic loss, the corresponding box plots are shown in Fig. 3.1(c). Since logistic loss is however also defined in the case of no clicks, in the left most plot we show the summaries including all the banner ads in the averages.

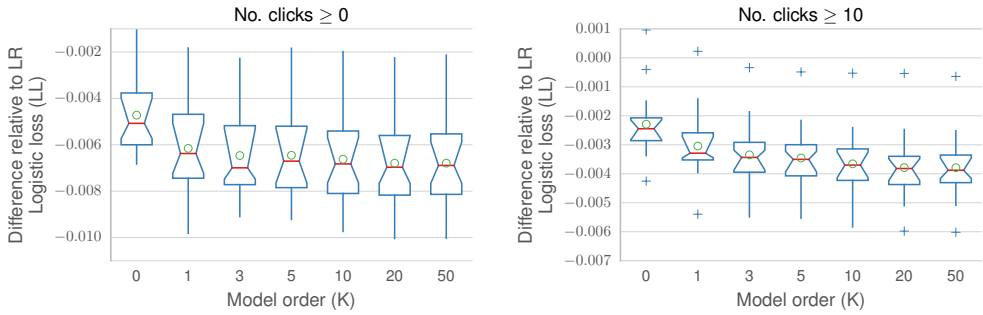
First of all, all the results agree that the LR+LFL$_K$, including $K = 0$, is better than using the LR model alone. The more tricky conclusion here, is whether higher model orders are advantageous. While each of the box plots in Fig. 3.1(b) and Fig. 3.1(c) certainly show this trend, not all of them are as significant. Specifically in the case of computing AUC scores based on all banner ads including those with very few clicks (left plot in (b)), the notches of all the models with $K > 0$ overlap with the one for $K = 0$, i.e., we cannot reject the null-hypothesis that these median values could be arising by chance (with 95% confidence intervals). For each of the other cases, however, for model orders $K = 20, 50$, we see that these do not overlap with the $K = 0$ case, hence statistically supporting that also the latent dimensions contribute significantly.



(a)



(b)



(c)

Figure 3.1: **(a)** Daily average AUC differences (i.e., increase) for LR+LFL$_K$ models using the optimal settings for different model-orders (colored lines) relative to the side-information model alone. **(b-c)** Box plots of the relative AUC (b) and LL (c) differences computed from each of the daily performances of the models on all 22 test days. The plot titles identify particular slices in test data based on number of clicks per banner. For more details, we refer to the source [26].

3.6 Summary

In this chapter, we have covered how a possible model for click-through rate prediction can be specified and elaborated on an extension which allows the addition of latent features in the form of the latent feature log-linear framework. Drawing on results from our contribution [26], we show that a sparse logistic regression model for click-through rate prediction can be extended and learned jointly with a latent feature log-linear model, and thereby obtain higher performances.

The latent feature model comes however at the cost of more hyper-parameters as well as the model order to tune. This process is counter-productive in its nature and thus is a hindrance to the adoption of the framework. It would be much more suited for adoption, if we could infer hyper-parameters and model order from data automatically, which we however do not think would be possible without modifications to the model specification and would likely also come at cost in terms of computation.

We do think that borrowing ideas from collaborative filtering to improve on methods on explicit features is a direction worthy of research. A collaborative filtering approach allows tuning in to recommender effects in data, e.g., abstract patterns of generalizable collective behavior, and in combination with models of explicit features, this has the added benefit of the possibility for making more reasonable predictions in settings where explicit features are missing.

The notion of latent variables as a representation of generalized collective behavior is also the theme of the next chapter. Here we show how a Bayesian generative model can be used for automatically clustering users and URLs into groups based on coherent structure of a network.

CHAPTER 4

Learning profiles of users and websites

In this chapter we turn to a discussion about building and learning models that can be used as additional features in supervised models, such as those discussed in the previous chapter, and in particularly in a logistic regression model such as the one used in production at Adform. We call this task *profiling* [24, 67] and it is an example of what is called *behavioural targeting* in computational advertising [33].

The motivation behind profiling is here mainly to enable better targeted ads for the impressions where there is little or no historical data with respect to a given metric. Take as an example a simple setup for click-through rate prediction, where the only features are the CookieId and the URL. If, for a specific pair (c_i, u_j), we denote the maximum likelihood click-through rate P_{ML}, defined as

$$P_{ML}(c_i, u_j) = \begin{cases} \frac{C_{ij}}{V_{ij}} & \text{if } C_{ij}, V_{ij} > 0 \\ ? & \text{otherwise} \end{cases} \tag{4.1}$$

where C_{ij} and V_{ij} counts the number of historical clicks and views for the pair (c_i, u_j), respectively, then the question is what value to use for "?". If it is the case that V_{ij} is incredibly large and $C_{ij} = 0$, then zero is probably not a bad estimation. However, it is a possibility that there simply has not yet been enough views to observe any clicks for the given pair.

If we now assume instead, that we have a model based on *views* only that can output that the CookieId is interested in, say cars, and that the current URL is one concerning cars, we now get another view of historic events where the user, now not restricted to c_i, is interested in cars and the URL, neither restricted to u_j, is also about cars, that we could base our estimator on. Presumably there are many users interested in cars and many websites that are about cars, hence it is much more likely that we have a good estimation of a CTR in that case. Hence, instead of "?", we could make the estimation based on the *segments* of the user and the URL instead, thereby hoping that the statistic generalizes.

With this as our main motivation, we therefore focus in this chapter on how to build segments based on browsing patterns in the past, and which we can test by measuring downstream performance in a click-through rate model using the segments as additional features.

This chapter overlaps with and supplements the publications Fruergaard et al. [29, Appendix D], Fruergaard and Hansen [27, Appendix E], and Fruergaard and Herlau [28, Appendix F] that are a part of this thesis.

4.1 Overview

In Adform there is an abundance of data recorded of the type (CookieId, URL), as well as other attributes available in the transaction, where an advertisement has been shown to a user identified by CookieId on a page identified by URL. For instance impressions from ad serving and real-time bidding requests (see Section 1.3), both labeled (e.g., click/no click) and unlabeled. For the vast majority of this data, it does not inform about the users clicking or conversion intent directly, but it is a possible source for engineering additional features of the user and the page, such as *interest segments*, that we can then use in a downstream click-through rate prediction model.

For the approach that we introduce in this section, we represent observations of the kind (CookieId= c_i, URL= u_j), $i = 1, \ldots, I, j = 1, \ldots, J$, as a *bipartite graph*. Recall, that a bipartite graph is a graph with two types of nodes, in this case users and URLs, which we refer to as the *modalities* of the graph, and that links are only between the modalities, not within. In this concrete example it means that a link exists between a user c_i and a URL u_j when that user has visited URL u_j. Furthermore, we note that direction is irrelevant, hence an undirected graph. In the following, we will refer to an undirected bipartite graph as it is realized by an *adjacency matrix* $\boldsymbol{A}$ with elements A_{ij} denoting the number of links between user c_i and URL u_j, i.e., the notation allows multi-edges, representing that the pair (c_i, u_j) may have been observed multiple times. If we binarize the adjacency matrix, hence discarding the frequency information in $\boldsymbol{A}$, we will refer to it instead as $\boldsymbol{B}$.

We then formulate the profiling task as a *co-clustering* problem [18, 36, 58]. In co-clustering, the goal is to group each modality into partitions of coherent structure based on the edges of the graph. Before we introduce a class of latent variable Bayesian models which we propose as a model of the User-URL graph, we give in the following a brief review of methods and their applicability to our problem.

We generally make a distinction between *latent feature* models and *latent class* models; the distinction, however in some cases a bit blurry, is whether a model assigns continuous valued weights of association to clusters, also known as soft-assignments, or if it assigns one or multiple discrete classes, referred to as hard-assignments.

Latent feature models

As briefly mentioned, we call a model that assigns continuous-valued vectors of cluster-associations to vertices a *latent feature* model. Disregarding that the input matrix in fact represents an adjacency matrix, this problem can be approached as an *unsupervised classification problem*, i.e., the true associations (as well as the cluster centroids) are unknown, yet we wish to find principal directions in data which maximize the complete-data likelihood function, which must be specified. In the following,

we introduce some techniques suitable for this task, all part of a class of methods known as *matrix factorization*.

Singular value decomposition

The singular value decomposition (SVD) [32] of a rank R matrix $\boldsymbol{A}$ is given as the factorization $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top = \sum_{i=1}^{R} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^\top$, where $\boldsymbol{U}$ and $\boldsymbol{V}$ are unitary matrices $\boldsymbol{U}^\top\boldsymbol{U} = \boldsymbol{V}^\top\boldsymbol{V} = \boldsymbol{I}$ and hold the left and right singular vectors of $\boldsymbol{A}$, respectively. The diagonal matrix $\boldsymbol{\Sigma}$ consists of the singular values, σ_i, of $\boldsymbol{A}$. By selecting only the K largest singular values of $\boldsymbol{\Sigma}$, i.e., truncating all other singular values to zero, the approximation $\tilde{\boldsymbol{A}} = \boldsymbol{U}\tilde{\boldsymbol{\Sigma}}\boldsymbol{V}^\top = \sum_{i=1}^{K} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^\top$ is obtained and is the rank K optimal solution to $\arg\min ||\boldsymbol{A} - \tilde{\boldsymbol{A}}||_2^2$. This truncation corresponds to disregarding the $R-K$ dimensions with the least variances as noise.

Non-negative matrix factorization

Non-negative matrix factorization (NMF) [51] is a matrix factorization similar to SVD, the crucial difference being that NMF decomposes into non-negative factors and imposes no orthogonality constraints. Given a non-negative input matrix $\boldsymbol{A}$ with dimensions $I \times J$, NMF approximates $\boldsymbol{A} \approx \boldsymbol{W}\boldsymbol{H}$, where $\boldsymbol{W}$ is an $I \times K$ non-negative matrix, $\boldsymbol{H}$ a $K \times J$ non-negative matrix, and K is the number of components. By selecting $K << min(I, J)$ one approximates $\boldsymbol{A}^{(I\times J)} = \boldsymbol{W}^{(I\times K)}\boldsymbol{H}^{(K\times J)} + \boldsymbol{E}^{(I\times J)}$, thereby discarding the residual (unconstrained) matrix $\boldsymbol{E}$ as noise.

NMF has achieved good empirical results as an unsupervised learning technique within many applications, e.g., for document clustering [10, 71, 74], visual coding [51], and bioinformatics [21]. In [16] multiplicative updates based on NMF are used for supervised learning of a click-through rate model in computational advertising. I.e., the approach is not applied for learning latent profiles, and the connection to NMF merely arises as a computational optimization from the non-negativity requirements on the weights of a linear model used as mean parameters in a Poisson observation model.

Least-squares matrix factorization

In its most general form, least-squares matrix factorization minimizes the objective $||\boldsymbol{A} - \boldsymbol{W}\boldsymbol{H}||_2^2$ with factors $\boldsymbol{W}$ and $\boldsymbol{H}$, subject to no constraints. If non-negativity constraints are added on $\boldsymbol{W}$ and $\boldsymbol{H}$, this corresponds to NMF with a least-squares objective. Regularization is often added to 1) control overfitting on a training data set and 2) impose structure on the latent factors. ℓ^2, ℓ^1, as well as *elastic net* [75], which is a trade-off between ℓ^1 and ℓ^2, are popular choices.

Maximum-margin matrix factorization

Maximum-margin matrix factorization (MMMF)[69] in contrast to the aforementioned least-squares models and SVD, aims to maximize the margin between the

columns and rows respectively of the matrices $\boldsymbol{W}$ and $\boldsymbol{H}$, while estimating the observed data $\boldsymbol{A} \approx \boldsymbol{W}\boldsymbol{H}$. It does so not by imposing low-rank on the reconstruction $\tilde{\boldsymbol{A}} = \boldsymbol{W}\boldsymbol{H}$, but rather imposes low-norm on $\tilde{\boldsymbol{A}}$. In practice this can be done by constraining $\boldsymbol{W}$ and $\boldsymbol{H}$ on their Frobenius norms as $\frac{1}{2}(||\boldsymbol{W}||_{\text{Fro}}+||\boldsymbol{H}||_{\text{Fro}})$ and optimizing an objective function other than least squares; see [69] for details.

Latent feature log-linear model

If the observed graph is binary, any of the above mentioned techniques will reconstruct the graph $\tilde{\boldsymbol{B}}$ with continuous values allowed outside the interval $(0, 1)$ (except non-negative methods, which do not reconstruct negative values, but are still not bounded by 1), which is an undesirable property, since this does not preserve the natural constraints of the input graph. In Section 3.3 we introduced the latent feature log-linear (LFL) and (without side-information) this solves this issue by passing the inner products between latent factors through the sigmoid, thus effectively projecting the unbounded continuous values back into the interval $(0, 1)$. The LFL model can be extended to general count matrices as well; for more information we refer to the original paper [54].

Latent feature stochastic blockmodels

Stochastic blockmodels (SBM) will be introduced in much more detail later in this chapter, however here we mention a couple of variants that are strikingly similar to the LFL model. In [5, 46] complex networks are modeled by inner products of continuous-valued latent features, that instead of being mapped through a sigmoid, act as the rate parameters in a Poisson-distribution, which is assumed being the stochastic process underlying observed links.

Latent class models

In latent class models the latent variables take discrete values. In the following we introduce *mixture models*, which are a type of latent class models and from those draw parallels to other techniques.

Mixture models

A mixture model has the form

$$p(\boldsymbol{x}|\boldsymbol{\pi}, \boldsymbol{\Phi}) = \sum_{k=1}^{K} \pi_k p(\boldsymbol{x}|\boldsymbol{\phi}_k), \tag{4.2}$$

where k indexes the *number of components* K and $\boldsymbol{\Phi} = \phi_1, ..., \phi_K$ denotes parameters of a probability density (or mass for discrete $\boldsymbol{x}$) function p. π_k is a mixing proportion and must satisfy $\sum_k \pi_k = 1$. We now introduce the latent class variables as a K-dimensional binary random variable $\boldsymbol{z}$ in which a particular element z_k is a 1 and the

rest are 0, i.e., a 1-of-K encoding. We then specify the marginal distribution as

$$p(\boldsymbol{z}) = \prod_{k=1}^{K} \pi_k^{z_k}, \tag{4.3}$$

i.e., the mixing coefficients π_k are the marginal probabilities of *success* (1) for the latent variables. By analogy Eq. (4.2) specifies the marginal distribution $p(\boldsymbol{x}) = \sum_{\boldsymbol{z}} p(\boldsymbol{z})p(\boldsymbol{x}|\boldsymbol{z})$, using that $p(\boldsymbol{x}, \boldsymbol{z}) = \prod_k \pi_k^{z_k} p(\boldsymbol{x}|z_k)^{z_k}$ and summing over all possible states $\boldsymbol{z}$.

Similarly, from Bayes theorem we obtain an expression for $p(z_k = 1|\boldsymbol{x})$,

$$p(z_k = 1|\boldsymbol{x}) = \frac{p(z_k)p(\boldsymbol{x}|z_k)}{\sum_{k'} p(z_{k'})p(\boldsymbol{x}|z_{k'})}. \tag{4.4}$$

We also refer to $p(z_k = 1|\boldsymbol{x})$ as the *responsibility* that component k takes in explaining the observation $\boldsymbol{x}$ [11, p. 432]. Recalling the sampling techniques introduced in Section 2.3, if we specify the likelihood $p(\boldsymbol{x}|\boldsymbol{z})$ and the prior $p(\boldsymbol{z})$ as follows

$$p(z_k|\boldsymbol{z}_{\setminus k}, \boldsymbol{x}) = \frac{p(\boldsymbol{z}_{\setminus k}|z_k)p(z_k)}{\sum_{k'} p(\boldsymbol{z}_{\setminus k'}|z_{k'})p(z_{k'})}, \tag{4.5}$$

we see that Gibbs sampling (see Eq. (2.73)) is indeed applicable to parameter inference for mixture models.

K-means clustering

One type of mixture model is the mixture of Gaussians, i.e., where we specify $p(\boldsymbol{x}|\boldsymbol{\phi}_k)$ as $\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$. It turns out when using the *expectation maximization* algorithm (see [11, p.443-444]) for inference on Gaussian mixtures, that with a shared variance parameter ϵ being shared between all components, i.e., $\boldsymbol{\Sigma}_k = \epsilon\mathbb{I}$ for all k, then in the limit $\epsilon \to 0$, the responsibilities $p(\cdots|\boldsymbol{x})$ go towards hard-assignments of clusters. This particular solution is equivalent to that assigned by *k-means clustering*; a particularly fast algorithm for clustering that we introduce in the following.

In k-means clustering, we keep track of K vectors that are the mean vectors, $\boldsymbol{\mu}_k$, of the points associated with each cluster, using hard-assignments. An objective function called the *distortion* is defined as

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} ||\boldsymbol{x}_n - \boldsymbol{\mu}_k||_2^2, \tag{4.6}$$

where z_{nk} denotes the k^{th} component of the one-of-K encoded indicator vector $\boldsymbol{z}_n$ representing the cluster currently assigned observation n. The algorithm proceeds in an iterative fashion by successively minimizing J with respect to z_{nk}, while keeping $\boldsymbol{\mu}_k$ fixed, and next with respect to $\boldsymbol{\mu}_k$ keeping z_{nk} fixed. This is run until convergence,

which is usually defined as when no more nodes change clusters in a single step of the algorithm.

K-means is a vector-similarity based approach and in the form we have just presented it, works in Euclidean distances. In principle any dissimilarity can be minimized instead and by using instead the "most centered" data point in each cluster as the *centroid* (substituting $\boldsymbol{\mu}_k$), we obtain instead the K-medioids algorithm [48].

Spectral clustering

Spectral clustering uses eigenvectors (i.e., the spectrum) derived from the graph to cluster entities into groups; either hierarchically finding K clusters by recursively splitting the input in two (e.g. [68]), or by computing some K eigenvectors, running k-means on the space spanned by these eigenvectors and finally mapping clusters back to entities in the original graph (e.g. [61]).

Spectral clustering, as opposed to k-means clustering and Gaussian mixtures, permit non-linear separation of clusters in input space. A variant of k-means called *kernel* k-means however extends k-means by non-linearly mapping data into a higher-dimensional feature space prior to running the algorithm, thus allowing for non-linearly discriminating clusters in input space. Work by Dhillon et al.[19, 20] shows that with particular choices of objective functions, weighted kernel k-means is equivalent to spectral clustering, thus allowing for a scalable "eigen-free" spectral clustering method based on k-means.

Neither k-means nor spectral clustering as presented above are formulated for bipartite graphs. A simple way to use either technique on bipartite graphs exists, however. If $\boldsymbol{A}$ is the adjacency matrix of a bipartite graph with I rows and J columns, then we can construct a symmetric matrix $\boldsymbol{W}$ with the structure,

$$\boldsymbol{W} = \begin{bmatrix} \mathbf{0} & \boldsymbol{A} \\ \boldsymbol{A}^T & \mathbf{0} \end{bmatrix}, \tag{4.7}$$

thus representing a unipartite graph with $I + J$ nodes, but where links are only present between the two modalities. Both k-means as well as spectral clustering can then be run directly on $\boldsymbol{W}$, but with the consequence that the resulting clusters "share" entities from both modalities. It is possible also to incorporate means for taking the special structure of $\boldsymbol{W}$ into account, see for instance Gu et al. [34].

Discussion

For any of the techniques outlined above, the application that we first of all have in mind at Adform, is to learn *extra* features about the users and URLs, that can be used in the supervised learning task for click-through rate prediction, which was the theme of Chapter 3. The conceptual system setup is sketched in Fig. 4.1. Since working with all-binary features in the click-through rate prediction model has some storage and computational advantages, we note that any of the latent feature representations of Section 4.1 are less preferable to the latent class representations in Section 4.1. In the

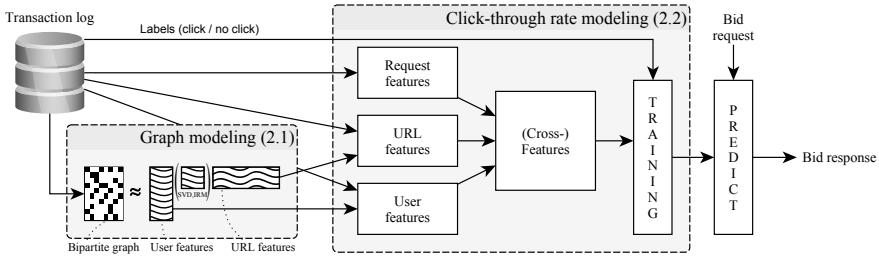


Figure 4.1: Proposed design of a pipeline for click-through rate prediction. The "wavy" matrices in the graph modeling part change depending on decomposition technique. The features passed on to the click-through rate modeling task from each of the different decompositions are vectors from the User and URL components, respectively. I.e., the User feature passed on is the transpose of the row-vector corresponding to the user of a training data observation and the URL feature is the column-vector corresponding to the URL of the observation. Figure from [27].

remainder of this chapter part of the contribution we make serves to investigate the consequences when using either latent feature or latent class models for constructing representations of profiles.

Between the concept of mixture models in general, k-means and spectral methods, that all give sparse and binary representations, the main advantage for mixture models is a flexible framework for modeling allowing, e.g., hierarchical Bayesian modeling techniques (the topic of the following section) that can be used to learn also the effective number of components needed *in each modality* separately. This is in contrast to k-means, spectral clustering, as well as any of the latent feature techniques, where both modalities will usually use the same number of components, which the user must specify.

Learning the parameters of a mixture model specified in a Bayesian framework, however, comes at the price of extra computational cost. In terms of processing speed, k-means and derivatives are often considered the fastest, yet we will show how to derive a blocked Gibbs sampler for Bayesian inference which is *scalable*, i.e., the cost per Gibbs sweep is linear in the number of realized edges in the input graph, and it can be very efficiently executed in parallel on modern *graphics processing units* (GPU).

4.2 Stochastic blockmodels

Stochastic blockmodels (SBM) were introduced by Holland et al.[41] building on ideas from White et al.[72] and Holland and Leinhardt[40]. SBMs are essentially mixture

models over adjacency matrices with *block structure*. By block structure, we assume that nodes in the graph represented by the adjacency matrix can be grouped into blocks and where edges between nodes can be described from the blocks to which nodes belong alone. Although White et al.[72] at least philosophically opens up to the possibility that nodes can belong to multiple groups, depending on the types of ties in a network, for now we stick to Holland and Leinhardt's[40] definition of SBM, where a node only belongs to a single group, which we call a *cluster*. Hence, for now we assume that the blocks of SBMs are *disjoint* or in other words, they are a *partitioning* of the graph. The "Stochastic" in SBM come from the definition that the links of the graph are assumed iid. generated under a probability distribution with parameters depending on the cluster assignments of the endpoints alone.

In this thesis we focus on a hierarchical Bayesian generative model for bipartite stochastic blockmodels, which we sketch as follows: Let $\boldsymbol{z}^{(1)} = (z_1^{(1)}, \ldots, z_I^{(1)})^T$, $\boldsymbol{z}^{(2)} = (z_1^{(2)}, \ldots, z_J^{(2)})^T$, and $\boldsymbol{\eta} \in \mathbb{R}_+^{(L \times M)}$ with elements η_{lm}, $l = 1, \ldots, L$, $m = 1, \ldots, M$. Then we write the model as,

$$\boldsymbol{\mu}^{(1)} \in [0,1]^{(l)}, \qquad \boldsymbol{\mu}^{(1)} | \alpha^{(1)} \sim \mathrm{Dir}(\alpha^{(1)}) \qquad \textit{cluster marginal} \tag{4.8a}$$

$$\boldsymbol{\mu}^{(2)} \in [0,1]^{(m)}, \qquad \boldsymbol{\mu}^{(2)} | \alpha^{(2)} \sim \mathrm{Dir}(\alpha^{(2)}) \qquad - \tag{4.8b}$$

$$\text{for } i = 1, \ldots, I, \qquad z_i^{(1)} | \boldsymbol{\mu}^{(1)} \sim \mathrm{Mult}(\boldsymbol{\mu}^{(1)}) \qquad \textit{cluster assignment} \tag{4.8c}$$

$$\text{for } j = 1, \ldots, J, \qquad z_j^{(2)} | \boldsymbol{\mu}^{(2)} \sim \mathrm{Mult}(\boldsymbol{\mu}^{(2)}) \qquad - \tag{4.8d}$$

$$\text{for } l \leq m, \qquad \eta_{lm} | \boldsymbol{\gamma} \sim g(\boldsymbol{\gamma}) \qquad \textit{link rate} \tag{4.8e}$$

$$\text{for } i < j, \qquad A_{ij} | \boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, \boldsymbol{\eta} \sim f(\eta_{z_i z_j}) \qquad \textit{link weight} \tag{4.8f}$$

This generative process can be stated as follows

(i) $\boldsymbol{\mu}^{(\cdot)} | \alpha^{(\cdot)} \sim \mathrm{Dir}(\alpha^{(\cdot)})$: Sample cluster marginals from a symmetric Dirichlet distribution parameterized by a single parameter $\alpha^{(\cdot)} > 0$ called the *concentration parameter*.

(ii) $z_{i(j)}^{(\cdot)} | \boldsymbol{\mu}^{(\cdot)} \sim \mathrm{Mult}(\boldsymbol{\mu}^{(\cdot)})$: Draw cluster assignment $1, \ldots, I(J)$ as *a single draw* from the Multinomial distribution with coefficients $\boldsymbol{\mu}^{(\cdot)}$, also sometimes referred to as the *Categorical* distribution.

(iii) $\eta_{lm} | \boldsymbol{\gamma} \sim g(\boldsymbol{\gamma})$: Generate intra- and inter-cluster link rates from a probability distribution g parameterized by $\boldsymbol{\gamma}$.

(iv) $A_{ij} | \boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, \boldsymbol{\eta} \sim f(\eta_{z_i z_j})$: Generate edges that are independently distributed according to f parameterized by the rates $\eta_{z_i z_j}$.

If we allow $L, M \to \infty$ we call the above "recipe" a *non-parametric* Bayesian model, where "non-parametric" refers to the number of parameters growing with the amount of training data. An interesting property which we will come back to, is that while the infinite dimensionality of the parameters may seem intractable, we *can*

actually make parameter inference in non-parametric Bayesian models, thus allowing to learn also the effective number of parameters from data.

By specifying f and g, the procedure Eq. (4.8) can be tailored to different types of networks. For the User-URL graph that we are interested in modeling in this chapter, we can either model a graph with counts of edges representing the number of times a specific (c_i, u_j) pair has been observed or we can truncate all the weights and analyze the binary graph. The former we refer to as a *counts network*, the latter a *binary network*.

Counts networks

If we wish to model a network $\boldsymbol{A}$ with integer weights on edges, we first note that we can use a Poisson distribution in place of f in Eq. (4.8f). Since, for practical purposes, we wish to keep conjugacy, we specify a Gamma distribution for g in Eq. (4.8e). I.e.,

$$\text{for } l \leq m, \qquad \eta_{lm}|\alpha, \beta \sim \text{Gam}(\alpha, \beta) \qquad \textit{link rate} \tag{4.9a}$$

$$\text{for } i < j, \qquad A_{ij}|\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, \boldsymbol{\eta} \sim \text{Pois}(\eta_{z_i z_j}) \qquad \textit{link weight} \tag{4.9b}$$

Binary networks

In the case that we analyze a binary network $\boldsymbol{B}$, i.e., where links only either exist or do not exist, we can change f in Eq. (4.8f) to a Bernoulli distribution, and as the Beta distribution is conjugate to the Bernoulli, we pick g in Eq. (4.8e) to be the Beta distribution. I.e.,

$$\text{for } l \leq m, \qquad \eta_{lm}|\beta^+, \beta^- \sim \text{Beta}(\beta^+, \beta^-) \qquad \textit{link probability} \tag{4.10a}$$

$$\text{for } i < j, \qquad A_{ij}|\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, \boldsymbol{\eta} \sim \text{Bern}(\eta_{z_i z_j}) \qquad \textit{link / no link} \tag{4.10b}$$

With $L, M \to \infty$, this particular formulation is also known as the *Infinite Relational Model* [49], however where the cluster assignments are sampled from the Chinese Restaurant Process (CRP, [2, 65]).

Note as well that the generative model Eq. (4.9) based on Poisson observations is for practical reasons often applied for binary networks. The error this introduces is small and for large, sparse networks vanishes as $\frac{1}{N}$, i.e., the number of multiedges that will be generated in a large network will be a small fraction of links only [5, 46].

Inference

Inference in the two models Eq. (4.9b) and Eq. (4.10b) is possible using Gibbs sampling, which we introduced in 2.3. Since in both models we specify g as the conjugate priors to f, it is possible to integrate out the $\boldsymbol{\eta}$ parameters and the resulting posterior will take the same form as the prior (Bishop[11]); in this case we would call it *collapsed* Gibbs sampling. However, if we do *not* integrate out the $\boldsymbol{\eta}$ parameters, the conditional distributions $p(z_i^{(1)}|\boldsymbol{z}_{\setminus i}^{(1)}, \cdot)$ and $p(z_j^{(1)}|\boldsymbol{z}_{\setminus j}^{(1)}, \cdot)$ become decoupled, i.e., they

can be sampled in parallel using a blocked Gibbs sampler. In Appendix B we show how these samplers are derived and also elaborate on how to do so for non-parametric versions.

We show in Appendix B how the conditionals for $\boldsymbol{z}_i^{(1)}$ and $\boldsymbol{z}_j^{(2)}$ (in the log-domain) decouples for each i and j respectively. With very large graphs, i.e., I and/or J being really large, sequential computation of these becomes intractable, especially since we will be performing these computations each Gibbs iteration. Due to the decoupling, they can however be computed in parallel. Yet even using modern CPUs having maybe 6-12 cores, this is still very time consuming. This motivates the use of modern graphics processing units instead, and is the focus of the following section.

GPU accelerated inference

Graphics processing units have, as their title suggests, traditionally been purposed for real-time graphics applications in computers, most notable in computer games. In recent years however, they are being taken advantage of in a wide range of computationally heavy applications where their special architecture can allow massive speedups. NVidia were the first GPU vendor to introduce in 2006 a complete stack, i.e., both hardware and software, for *general purpose* GPU computing (GPGPU) [63], thereby lessening the burden of tapping into the GPUs processing powers. The platform is called CUDA and is the leading GPGPU toolkit, both in terms of features and performance. As a part of this thesis, we have implemented the most taxing computations of the blocked Gibbs samplers derived in Appendix B, namely the inference of the assignment variables $\boldsymbol{z}_i^{(1)}$ and $\boldsymbol{z}_j^{(2)}$, in CUDA. In this section we present an overview of the requirements and challenges for this sampler to run efficiently using CUDA.

While higher level abstractions exist to leverage the speed of GPUs, these are based on *floating point* matrix and vector routines. In SBMs the assignment vectors $\boldsymbol{z}^{(1)}$ and $\boldsymbol{z}^{(2)}$ are most conveniently stored as binary matrices where each row is a one-of-K encoded variable, in which case they can be used in matrix-vector and matrix-matrix linear algebra routines. However, it is this particular structure of the assignment matrices that makes our GPU implementation faster, which we will come back to shortly.

In Fig. 4.2 a diagram is shown of the typical workflow in GPGPU. The process consists of roughly four steps indicated in the figure. Typically a program will switch between executing code on the CPU and only offloading heavy computations to the GPU, thereby switching between host and GPU code, which is commonly known as *heterogeneous computing*. An important aspect here is the copying of contents from main memory to the GPU memory and vice versa, once processing on the GPU is complete. Since this is *slow* relative to the number of clock cycles the GPUs are idling while copying to and from main memory, we need to keep this to a minimum.

A second consideration is taking into account the general architecture of the GPU, when designing the software that is executed there. CUDA GPUs consist of multiple *symmetric multiprocessors* (SMP) which are capable of executing multiple threads

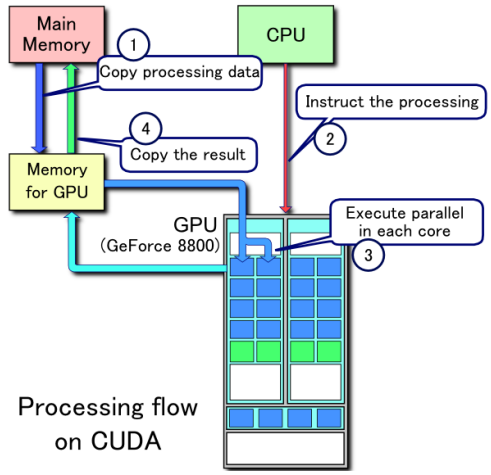


Figure 4.2: An overview of a typical GPU processing flow: 1) Copy data from main memory to GPU memory, 2) instruct the GPU to run a *kernel* (a special GPU function) on data, 3) processing happens in parallel on multiple cores on the GPU, and 4) the processed data is copied from the GPU back to main memory. *source: Tosaka [CC-BY-3.0 (http://creativecommons.org/licenses/by/3.0)], via Wikimedia Commons.*

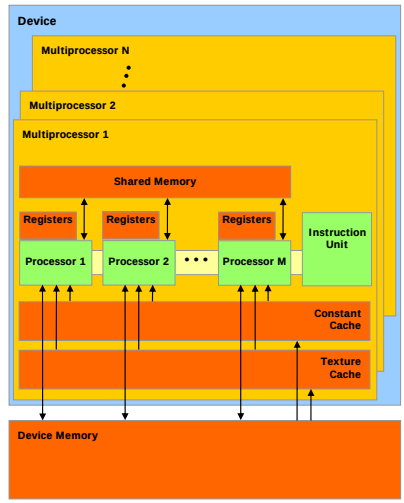


Figure 4.3: A set of SMPs (symmetric multiprocessors) with on-chip shared memory. *source: NVidia Cuda C Programming Guide http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf* (Last visit 11/21/2014).

simultaneously, subject to the same instruction being executed in each thread; an execution model called *single instruction multiple data* (SIMD). All SMPs can access a device memory, being the largest memory on the GPU. In each SMP there is a shared memory, a texture cache and a constant cache, which each of the processors in the same SMP can access, and which is orders of magnitude faster than accessing the device memory. This architecture is illustrated in Fig. 4.3.

Hence, in order to take advantage of GPU processing, we identify three main criteria:

(i) Copies to and from main memory should be as little and infrequent as possible.

(ii) Data processing on the GPU must be carefully designed to take advantage of SIMD execution.

(iii) Whenever possible the access to device memory should be kept to a minimum by copying data into shared memory for processing.

Although the points (ii) and (iii) in the above are not trivial and require lots of fine-tuning, a detailed discussion of how we implement the samplers in CUDA is out-of-scope of this thesis. What is important is that the update to each $z_i^{(1)} \in \boldsymbol{z}^{(1)}$ can be performed in parallel for all i, i.e., there are no data-dependencies. The same holds for $\boldsymbol{z}^{(2)}$. Therefore these can be efficiently implemented for SIMD execution.

What we will focus on instead, is how we handle big graphs that exceed device memory (on a single GPU) and thereby addressing point (i). We roughly describe the memory management of our current implementation but also how we envision an extension which can distribute a graph over several GPUs, thereby obtaining much more efficient parallelism.

At this point it makes sense to focus on the particular updates to a single one of the assignment vectors $\boldsymbol{z}^{(1)}$ and we will focus on the Gibbs update for the generative model Eq. (4.9b) for count networks; for the derivation see Appendix B.

In Matlab syntax a few lines implement the Gibbs update to $\boldsymbol{z}^{(1)}$ for regular CPU computation,

```
% A=sparse(I,J), Z2=sparse(M,J) (binary)
% eta=dense(L,M), log_mu1=dense(L,1)
AjZt=A*Z2';
Nmz2=sum(Z2);                                % size: (M,1)
etaNmz2=eta*Nmz2;                            % size: (L,1)
logL_Zj=log_mu1 * ones(1,I) ...              % (repeats log_mu1 in I columns)
        + log(eta) * AjZt'  ...
        - etaNmz2 * ones(1,I);               % (repeats etaNmz2 in I columns)
pz=exp(logL_Zj-ones(1,L)*max(logL_Zj));      % (repeats the max in L rows)
pz=pz./(ones(1,L)*sum(pz));                  % (repeats the sum in L rows)
pz_cs=cumsum(pz);
indicZ1=(M+1-sum(ones(1,L)*rand(1,I)<pz_cs))';
Z1=accumarray([indicZ1,(1:I)'],ones(I,1),[M,I],[],[],true);
```

Briefly, regarding performing all of the above operations with high level floating point linear algebra GPU routines, it is particularly performing the sparse matrix multiplication `A*Z2'` which due to the special structure of $\boldsymbol{Z}^{(2)}$ can be implemented entirely without multiplication thus only involving summation, i.e., $\boldsymbol{Z}^{(2)}$ indicates which elements in $\boldsymbol{A}$ to sum (if they are non-zero). Furthermore, implementing lines 6-10 in custom routines, allows extensive use of shared memory, which a high level routine cannot achieve.

The above (simplified) code snippet reveals some basic memory requirements for the sampler and we identify the largest: First of all $\boldsymbol{A}$ is an $I \times J$ sparse matrix, i.e., with memory requirements being linear in the number of nonzero elements. Second, we see that as intermediate results, we need a dense $M \times I$ matrix `AjZt'` and a dense $L \times I$ matrix `logL_Zj` in device memory. Regardless of the available device memory, either one of these matrices (or all) can grow too big to hold at once in a single GPU, thus must be handled. The solution is to slice $\boldsymbol{A}$ along the I dimension, since for each update to $z_i^{(1)}$, a single row of `Z1` in the code above, computation involves only the corresponding row of $\boldsymbol{A}$; the same reason the computations parallelize. By slicing $\boldsymbol{A}$ along I we thus control the size of the sparse as well as the dense matrices that needs to be held in device memory. For updating $z_j^{(2)}$, the methodology is the same, we just slice A along J. In that way, our sampler will handle arbitrarily large input graphs.

The downside is that in addition to communicating updated cluster assignment matrices $\boldsymbol{Z}^{(1)}$ and $\boldsymbol{Z}^{(2)}$ to and from host memory each Gibbs sweep, each accompanying slice of $\boldsymbol{A}$ needs to be copied to the device memory every time as well. This substantially increases the host-to-device memory copying. For our current implementation of the sampler, this is what we do.

In order to reduce this memory overhead we envision distributing each Gibbs sweep across multiple GPUs. The extension is relatively straight forward: Put a slice along I and one along J of $\boldsymbol{A}$ for which we can make a Gibbs sweep without exceeding device memory on each GPU. Then $\boldsymbol{A}$ is only copied to device memory once in the initialization. This of course requires as many GPUs as there are slices of $\boldsymbol{A}$, but if that is not possible, then at least for each time we double the number of GPUs, we will approximately half the time spend copying $\boldsymbol{A}$ to devices, since this will take place in parallel.

Extensions

A couple of extensions are possible with the SBM framework as it is formulated in Eq. (4.8), (4.9), and (4.10) and we wish to briefly mention these.

Degree-correction

It has been suggested one can include degree correction into the formulation of SBMs for count data (i.e., Poisson observations), which models the within-group degree heterogeneity which is found in many real-world networks and helps with the identifica-

tion of latent structure [39, 46]. In our Bayesian framework the same can be achieved with the assumption of additional Gamma-distributed parameters $\boldsymbol{\psi}^{(1)} = \psi_1^{(1)}, ..., \psi_I^{(1)}$ and $\boldsymbol{\phi}^{(2)} = \psi_1^{(2)}, ..., \psi_J^{(2)}$ and then in place of Eq. (4.9b), we specify

$$\text{for all } i, \qquad \psi_i^{(1)} | \pi^{(1)}, \rho^{(1)} \sim \text{Gam}(\pi^{(1)}, \rho^{(1)}) \qquad \textit{degree } i \tag{4.11a}$$

$$\text{for all } j, \qquad \psi_j^{(2)} | \pi^{(2)}, \rho^{(2)} \sim \text{Gam}(\pi^{(2)}, \rho^{(2)}) \qquad \textit{degree } j \tag{4.11b}$$

$$\text{for } i < j, \quad A_{ij} | \psi_i^{(1)}, \psi_j^{(2)}, \boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, \boldsymbol{\eta} \sim \text{Pois}(\psi_i^{(1)} \eta_{z_i z_j} \psi_j^{(2)}) \qquad \textit{link weight}, \tag{4.11c}$$

and with the other parameters specified as earlier. With conjugate priors as per the Gamma distributions, this remains tractable. We do not show the resulting sampling distributions here, but the important thing to realize is, that the efficient sampling of $\boldsymbol{z}^{(1)}$ and $\boldsymbol{z}^{(2)}$ on the GPU remains possible i.e., they can be sampled independently, in parallel. Our implementation also has this functionality, with the slight downside that we introduce four new hyper-parameters and which brings us to the following section.

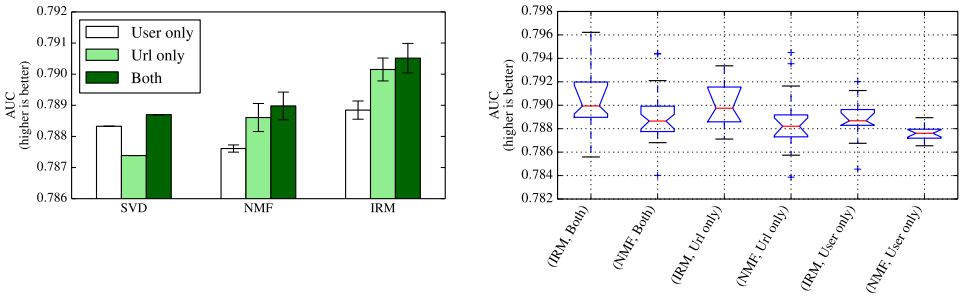
Inference of hyper-parameters

Our current implementation of the aforementioned specializations of stochastic block-models requires the hyper-parameters to be specified by the user. An extension that is possible however is to place non-informative priors on these and infer them using random-walk Metropolis steps. Although the acceptance rate for Metropolis-Hastings with random proposals can be very low, the steps are computationally cheap enough that even multiple steps each sweep comes generally at a low cost compared to the sampling of $\boldsymbol{z}^{(1)}$ and $\boldsymbol{z}^{(2)}$ [28, 39]. It should be said however, that we have not attempted this with the GPU accelerated Gibbs sampler presented herein, so a future exercise would be to reaffirm this approach in our specific setup.

4.3 A comparison of profiles for click-through rate prediction

In this section we introduce and discuss the two contributions Fruergaard et al. [29] and Fruergaard and Hansen [27], both part of this thesis and attached in Appendix D and Appendix E, respectively. Since [27] is a continuation of the work presented in [29] and offers a more thorough analysis as well as revises our previous conclusion, we focus on the results from [27].

In Fruergaard and Hansen [27] we test the performances of different representations for profiles in a system as the one depicted in Fig. 4.1. We analyze the binary graph with adjacency matrix $\boldsymbol{B}$ for $I = 99,854$ users and $J = 70,436$ URLs sampled from Adform's ad transaction logs over a period of 29 days and where a link $B_{ij} = 1$ signifies that user i was observed viewing URL j at least once in the dataset. The representations that we investigate are SVD and NMF (introduced in Section 4.1), as



(a) **(b)**

Figure 4.4: (a) Average click performances in AUC for the different decompositions (at optimal settings) and using only user (white), only URL (light green), or both profiles (dark green) as predictors. The error bars show ± one standard deviation from the mean based on 25 random restarts. (b) Box plots of the 25 random restarts for IRM and NMF AUC performances. Box notches ("slanted" edges) are 95 confidence interval estimates of the medians from 10k sample bootstrapping. Figures from Fruergaard and Hansen [27].

well as the IRM, which is a specialization of a Bayesian stochastic blockmodel with Bernoulli observations, as we also mentioned in Section 4.2.

The particular choice of those graph decomposition techniques is grounded in their fundamentally different representations. I.e., the SVD of $\boldsymbol{B} \approx \boldsymbol{U}\tilde{\boldsymbol{\Sigma}}\boldsymbol{V}^T$ where $\tilde{\boldsymbol{\Sigma}}$ is the diagonal matrix of the K largest singular values, puts continuous values in the loadings of $\boldsymbol{U}$ and $\boldsymbol{V}$. These loadings we then *join* on the user ids ($\boldsymbol{U}$) and the URLs ($\boldsymbol{V}$), respectively, to use as features in a click-through rate prediction model. The NMF on the other hand puts only *non-negative* values in the loadings for each modality and is known to produce sparse components, i.e., set to zero many of the values. These loadings are however still continuous valued. Finally, the IRM model makes a *hard-assignment* (clustering) in each modality, which can be represented as one-of-K encoded binary vectors, i.e., this is the "most compact" representation, both since it is the "most sparse" as well as binary.

The latter representation is thus most in concordance with the properties and constraints for Adform's click-through rate prediction task (see Section 3.1) and it is therefore interesting to investigate if this representation is a trade-off, a benefit, or simply status-quo in comparison with the other two representations.

In Fig. 4.4 we show some of the main results of Fruergaard and Hansen [27]. The categorization of "URL only", "User only", and "Both" here refer to whether only the URL loadings, only the user loadings, or both loadings for each respective model are used as additional features to the click-through rate prediction model.

In the first plot, Fig. 4.4(a), the results have been arranged in such a way, that it is more easily seen that for the two best performing representations, NMF and IRM, there is a higher gain when using only the URL loadings (light green) compared to using only the user loadings (white). Furthermore, the gain from URL loadings only to using both loadings as features (dark green) is more subtle and is just within the errorbars, here representing standard deviations of the mean.

In the second plot, Fig. 4.4(b), a boxplot is used to compare between the IRM models and NMF models only. The "notches" of the boxes are bootstrap (10k sample) estimates of 95% confidence intervals for the medians (red lines). I.e., non-overlapping notches indicate the medians are *different* under the null-hypothesis with $p < 0.05$. In the paper we also report a p-value of $p = 0.02$ for a two-tailed t-test between the results for (IRM,Both) vs. (NMF,Both), i.e., consistent with the non-overlapping notches in the figure. It is this latter result in particular, which lead us to the conclusion that profiles based on the compact representation of IRM can be superior to those based on NMF and SVD.

Of course these results we report for performance measured on a single day, so a further study could be to study whether these results are consistent over time, i.e., with more independent test sets. Since we submitted the paper [27] we have conducted a test in Adform comparing just the IRM model features in addition to the current set of features used in the production model. These results are show in Fig. 4.5 and confirm consistent improvements, albeit over a period of just 6 days.

A different aspect which we only briefly touch upon in the paper [27], is the time to run inference for the IRM model. Here we of course utilize the GPU acceleration scheme that we discussed in Section 4.2 and are thereby able to speed up the inference by a factor of approximately five times. As we also mentioned earlier, this number we expect could significantly higher, if we limit the copying between the host memory and the device memory. In the case of the graph we study in the paper, this is small enough to permanently reside in the device memory, which is a property that we do not exploit. Specifically our current implementation fully allocates, copies and deallocates the graph *twice* per Gibbs sweep, once for updating $\boldsymbol{z}^{(1)}$ and once as its transpose for $\boldsymbol{z}^{(2)}$. While not being particularly efficient for small graphs, this allows the implementation to work with arbitrarily large input graphs, where the graph is split in either dimension and chunks that fit in device memory are sequentially processed on the GPU.

4.4 A simple and scalable approach to overlapping community detection

In our submission Fruergaard and Herlau [28] which can also be found in Appendix F we venture into an interesting elaboration of a model for overlapping community detection and with a remarkable similarity to the stochastic blockmodeling techniques we have been discussing so far. This model which we call *multiple-networks stochastic blockmodel* (MNSBM), we discuss in the present section.

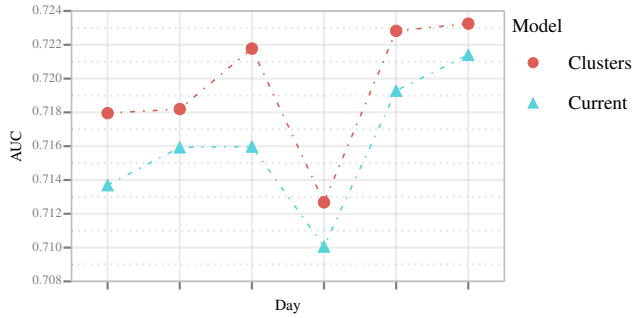


Figure 4.5: Daily average AUC score over all banners with non-zero clicks, comparing the production model features (Current) with a model using also IRM model features for both URLs and users (Clusters).

Stochastic blockmodels are in a sense limited by the fact that they assume disjoint partitions of a network. For instance within the field of *community detection* this limitation is pointed out as a weak point [50] and consequently there exist numerous approaches that relax this assumption in order to model overlapping community structure. These methods can be generally classified as (i) models which assume instead of partitions a multi-set of the vertices, i.e., several non-empty and potentially overlapping subsets of vertices [50, 57, 59, 64] and (ii) models that make a continuous relaxation of the assignments vectors, such that each vertex is associated a continuous-valued vector where each parameter models the degree of the vertex' membership to a "community" [1, 5, 12, 46, 54]. In either of these approaches, the difficulty arises that the probability of two vertices linking to each other is a function of all the blocks the two vertices belong to or are associated with.

One of the earlier formulations of stochastic blockmodels, although they were not known as such at the time, is White et al. [72], and we argue they make a subtle distinction in their treatment of *networks* as opposed to the more popular approach introduced by Holland et al. [41]. White et al. says basically that communities of a population can only be understood from knowing many different types of ties (i.e., edges), but that for a network for *one type of tie*, the vertices can be partitioned into distinct sets. This distinction inspired our interpretation: That observed networks are *complex* due to multiple networks of *different types of edges*, but each with simple (i.e., block) structure, being aggregated by an unknown function over networks. An example could be the networks of friendships we observe from an online social network; suppose the friendships can be classified into several categories such as family, friends, work colleagues, classmates, etc. It is naturally to assume these distinct types of edges overlap, which makes the composited picture of the network difficult to model. If we know however the separate networks for each type of ties, it is possible as White et al. motivates, that the simplifying assumption of grouping vertices into disjoint subsets *for each network* (such as in SBMs) is sufficient.

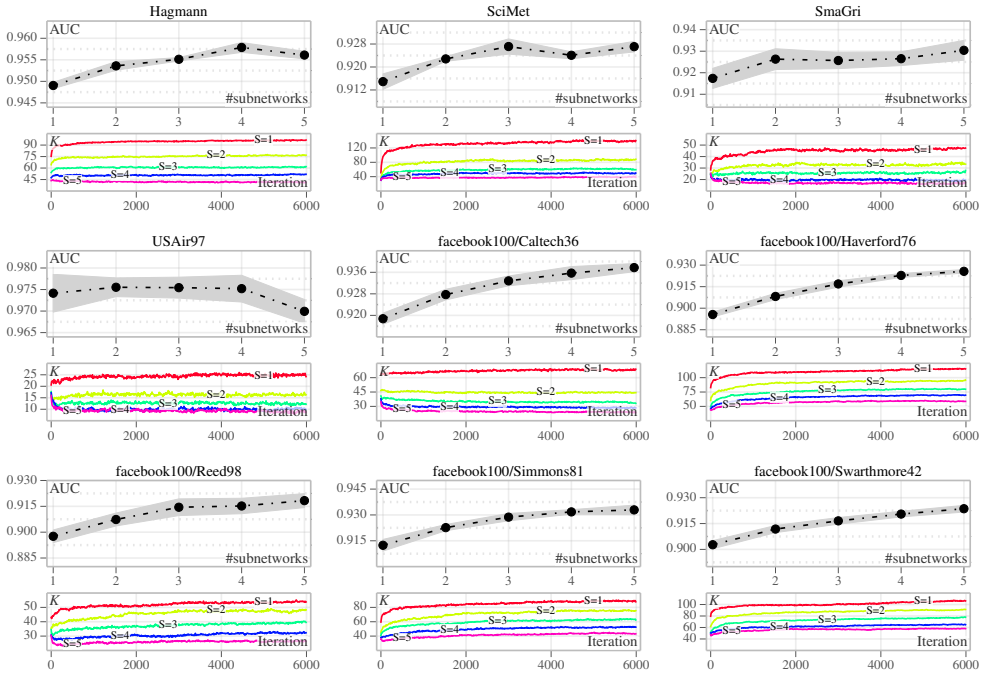


Figure 4.6: MNSBM results for each of the networks introduced in [28, p. 7]. For each network, MNSBM is run with $S = \{1, 2, 3, 4, 5\}$, i.e., varying number of subnetworks, and for 6,000 iterations. In the upper plots the average AUC link prediction scores are shown as a function of S. The AUC score for a single chain is computed as the average of the predictions from every 10^{th} iteration of the last half of the chain, discarding the first 3,000 as burn-in. The averages are based on 5 random restarts and 5% edges and non-edges picked randomly for testing. Shaded regions represent the standard deviation of the mean. In the lower plots we show the average number of detected components per subnetwork as a function of iterations, with each line representing an MNSBM with a different S. The averages are computed similarly to the AUC scores. Figure from [28]

We see this as a motivation for introducing the MNSBM, which essentially is an aggregation over stochastic blockmodels modeling edges as Poisson distributed random variables, but where each SBM is only observing a subset of the observed network, so as to emulate different types of edges. The resulting inference problem is thus the joint problem of splitting the observed edges between subnetworks and identifying (block) structure in each subnetwork. We derive an efficient Gibbs sampler for this model and note that the inference problem for each subnetwork can be sampled independently allowing for parallel inference.

In Fig. 4.6 we show the results on real-work network data sets from [28]. From the AUC scores, we see an overall increase in link prediction performance as we allow more subnetworks in the MNSBM model. The only exception is USAir97, which is also the smallest network in the benchmark. Either USAir97 does not exhibit overlapping structure or the small size of the dataset is an inhibiting factor.

Furthermore, each Gibbs sweep in a subnetwork scales as $\mathcal{O}(EK^2)$, where E is the number of edges and K is the number of blocks in that subnetwork. As can also be seen in Fig. 4.6, the average numbers of inferred blocks per subnetwork decrease as more subnetworks are added to the model, thus lowering the quadratic term in the computational complexity.

4.5 Summary

We have so far introduced stochastic blockmodels for the bipartite graph of users and URLs and seen that the partitions that we obtain are beneficial also to the click-through rate prediction problem, when they are used as additional features.

Besides the applications for click-through prediction, there are many other possible use-cases for the partitions. The fact that the profiles from the SBM-family are partitions mean that they are easy to store and easy to inspect. For instance, we envision the user and URL segments can be used also for exploratory analysis and selection even by the end users of Adform's system (typically a *campaign planner*). There are other machine learning tasks within Adform where they would also make sense as extra features. As one example, there is the task of expanding a pool of users with known demographics to a larger population, where user segment from an SBM could be a good predictor.

The integration of stochastic blockmodels as a continually updated automated segmentation model would still need to overcome a few obstacles. First of all, we would need a mechanism for new users and new URLs to be added in the model, without having to start all over; folding-in/imputation would be a feasible strategy. Another challenge would be if, say "URL cluster 01" from one model (or at some instance in time), suddenly becomes "URL cluster 20". First of all, how can we detect that happening and how do we then cater for it? One (simple) option would be to compute distances or correlations between the partitions of different models and matching up those that are close or very correlated. This matching could also be on the "cluster loadings", i.e., on the rows and columns of the $\boldsymbol{\eta}$ matrix instead.

These challenges are not limitations of the SBM framework, rather they would need to be addressed regardless of the methodology, except if a model can be specified to directly incorporate sufficient mechanisms to deal with those. Hence, we believe these challenges can be addressed both as merely tasks of engineering or as opportunities for extending the modeling framework.

One of the main limitations of SBMs is the partitioning into disjoint sets of entities. At the same time, this makes the parameters easier to infer, so there is a trade-off between complexity and feasibility. We addressed however one possible extension of the SBM framework, the multiple-networks stochastic blockmodel, which is effectively a multiple-membership model that avoids extra complexity on the groups or their interactions, thereby keeping the sampler simple and efficient. We have not yet optimized the implementation of the sampler for MNSBM for really large networks, but the same techniques that we introduce for SBMs with Poisson observations for speeding up inference using GPUs, apply here as well. MNSBM is basically just running multiple SBM samplers with Poisson observations (which we already have) in parallel and with a little extra coordination needed for the networks.

Regardless of the specific model choice or methodology, we think that automated segmentation based on the (user, URL) graph is a means to explore and monetize one of the bigger (if not biggest) sources of information which gives Adform a competitive advantage.

CHAPTER 5

Conclusion

In this thesis we have investigated machine learning methods with applications in performance-optimized web banner advertising, and we have demonstrated our results on applying them in the particular case of click-through rate prediction.

In our first contribution [26], we reviewed a hybrid model for click-through rate prediction, which combines sparse logistic regression with another probabilistic model using matrix factorization as latent features of Internet domains and ads. We have contributed details, both for modeling as well as experimentation, which are not found elsewhere and shared our experiences applying this model to real-world data from Adform. We tested the model and reported performances of multiple versions of the hybrid model with different dimensionality of the latent dimension over a period of 22 days, and found that using the hybrid model with several latent features improves click-through rate prediction compared to sparse logistic regression alone.

Next, we motivated the use of behavioral "profiles" based on browsing history as predictors in click-through rate modeling. The profiling task we formulated as a bipartite graph decomposition problem, where the two types of entities in the graph are users and URLs, respectively, and links represent the number of visits a user had on a given URL. While this graph does not contain information about clicks, it has the advantage of being much denser than a graph of only the clicks, and this makes it possible to learn profiles for a much larger population. Our contribution is to model browsing history graphs using non-parametric Bayesian stochastic blockmodeling. We showed how Gibbs samplers can be specified for both binary and counts graphs and how these can be efficiently implemented using GPGPU computing to accelerate the inference. In our contributions [27, 29], we analyzed real-world browsing history graphs from Adform using the stochastic blockmodels. We also computed profiles based on non-negative matrix factorization and singular value decomposition of the same graph. By evaluating the performances of using the different representations as additional predictors in a click-through rate model, we concluded that the profiles from stochastic blockmodels are better alternatives, both in terms of performance, but also due to their representation being sparse and binary.

The stochastic blockmodels that we introduced for modeling the browsing history graph of users and URLs are in general limited by the fact, that they impose single memberships and that links are "explained" based on the groupings of the two end-points alone. Our last contribution [28] introduced the multiple-networks stochastic blockmodel as a novel approach for allowing multiple memberships. This model, we argued, has several advantages to other multiple-membership models, most notably

in terms of its simple specification as well as the possibility for efficient parallelized inference. We tested this model on a number of real-world network datasets, where using more subnetworks showed improved link-prediction performances, compared to the single-network stochastic blockmodel. In terms of applications in Adform, it would be particularly interesting in the future to apply this model in the profiling task.

Overall in this thesis, we believe to have contributed valuable research in machine learning for computational advertising. The results we present for improvements in click-through rate prediction on Adform's data are generally positive and statistically significant. With that being said, we think that it is relevant to consider briefly the relative performance differences that we have reported: For both the hybrid model in Chapter 3 and the IRM profiles in Chapter 4, our relative improvements in terms of AUC are roughly between 0.002-0.004 and it might not seem like a lot. As [52] also points out, for click-through rate data, two things need to be taken into consideration: 1) Simple models based on baseline predictors, such as just the URL and position of the ad slot, achieve relatively high AUC scores, in our experience often just 0.01-0.02 below that of the full model, and 2) even predicting probabilities perfectly, the AUC would still be perhaps only 0.85 due to the inherent uncertainty of clicks. The practical implication is, that the "dynamic range" of the AUC where we are comparing models, is very "narrow", so even small changes matter.

This opens up to a more broad discussion about different evaluation metrics as well as tools for data and model visualization. Instead of measuring AUC and LL on "offline" data, candidate models can be run as live tests in an A/B testing framework, thus allowing also measuring the more quantifiable measures related to the clients needs, i.e., effective cost-per-click, and comparing that to both the target cost-per-click as well as making statistical tests and model comparison on these direct measures of performance. It is our ambition that we can do so in the near future. For qualitative and visual exploration, we think that stochastic blockmodels are well suited, due to their simple construction. Specifically, we envision a system where user-specified target measures (e.g., number of impressions, clicks, conversions, etc.) from past performance data can be logically joined with the segments from a stochastic blockmodel, which can thereby be ranked and scaled to visualize the relative targeting strengths.

APPENDIX A

Logistic loss gradient

Consider the logistic loss function given by,

$$L(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{\omega}) = -(\sum_{n=1}^{N} y_n \log(\sigma(\boldsymbol{\omega}^T \boldsymbol{x}_n)) + (1 - y_n) \log(1 - \sigma(\boldsymbol{\omega}^T \boldsymbol{x}_n))), \tag{A.1}$$

where $\boldsymbol{y} = (y_1, \ldots, y_N)^T$ is the binary response vector, $\boldsymbol{X} = (\boldsymbol{x}_i, \ldots, x_N)$ is the $P \times N$ feature matrix with a single element denoted by X_{pn}, and $\boldsymbol{\omega} = (\omega_1, \ldots, \omega_P)^T$ the coefficient vector.

We derive the derivative with respect to a single coefficient ω_p:

$$\frac{\partial L(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{\omega})}{\partial \omega_p} = -(\sum_{n} y_n X_{pn} \frac{\sigma'(\boldsymbol{\omega}^T \boldsymbol{x}_n)}{\sigma(\boldsymbol{\omega}^T \boldsymbol{x}_n)} - (1 - y_n) X_{pn} \frac{\sigma'(-\boldsymbol{\omega}^T \boldsymbol{x}_n)}{-\sigma(\boldsymbol{\omega}^T \boldsymbol{x}_n)}), \tag{A.2}$$

where we have used that $1 - \sigma(\boldsymbol{\omega}^T \boldsymbol{x}_n) = \sigma(-\boldsymbol{\omega}^T \boldsymbol{x}_n)$. We then use $\sigma'(\boldsymbol{\omega}^T \boldsymbol{x}_n) = \sigma(\boldsymbol{\omega}^T \boldsymbol{x}_n)(1 - \sigma(\boldsymbol{\omega}^T \boldsymbol{x}_n))$,

$$= -(\sum_{n} y_n X_{pn} (1 - \sigma(\boldsymbol{\omega}^T \boldsymbol{x}_n)) - (1 - y_n) X_{pn} \sigma(\boldsymbol{\omega}^T \boldsymbol{x}_n)) \tag{A.3}$$

Now, since $y_n \in \{0, 1\}$ it follows that $y_n \sigma(\cdot) + (1 - y_n)\sigma(\cdot) = \sigma(\cdot)$, hence the above reduces to

$$= -\sum_{n} X_{pn} (y_n - \sigma(\boldsymbol{\omega}^T \boldsymbol{x}_n)). \tag{A.4}$$

Now we see the gradient $\nabla_{\boldsymbol{\omega}}$ can be efficiently computed using matrix-vector products,

$$\nabla_{\boldsymbol{\omega}} = \boldsymbol{X} \cdot (\boldsymbol{y} - \sigma(\boldsymbol{X}^T \boldsymbol{\omega})) \tag{A.5}$$

□

APPENDIX B

Blocked Gibbs sampling for stochastic block models

In this appendix we walk through the derivations for blocked Gibbs samplers of the stochastic block models discussed in Section 4.2.

First, let $\boldsymbol{z}^{(1)} = (z_1^{(1)}, \ldots, z_I^{(1)})^T$, $\boldsymbol{z}^{(2)} = (z_1^{(2)}, \ldots, z_J^{(2)})^T$, and $\boldsymbol{\eta} \in \mathbb{R}_+^{(L \times M)}$ with elements η_{lm}, $l = 1, \ldots, L$, $m = 1, \ldots, M$. The generative model of the SBM for count networks, *with cluster sizes fixed to L and M* respectively, is

$$\begin{aligned}
\boldsymbol{\mu}^{(1)} &\in [0,1]^l, & \boldsymbol{\mu}^{(1)}|\alpha^{(1)} &\sim \mathrm{Dir}(\alpha^{(1)}) & \textit{cluster marginal} \quad & \text{(B.1a)}\\
\boldsymbol{\mu}^{(2)} &\in [0,1]^m, & \boldsymbol{\mu}^{(2)}|\alpha^{(2)} &\sim \mathrm{Dir}(\alpha^{(2)}) & \text{-} \quad & \text{(B.1b)}\\
\text{for } i &= 1, \ldots, I, & z_i^{(1)}|\boldsymbol{\mu}^{(1)} &\sim \mathrm{Mult}(\boldsymbol{\mu}^{(1)}) & \textit{cluster assignment} \quad & \text{(B.1c)}\\
\text{for } j &= 1, \ldots, J, & z_j^{(2)}|\boldsymbol{\mu}^{(2)} &\sim \mathrm{Mult}(\boldsymbol{\mu}^{(2)}) & \text{-} \quad & \text{(B.1d)}\\
\text{for } l &\leq m, & \eta_{lm}|\alpha, \beta &\sim \mathrm{Gam}(\alpha, \beta) & \textit{link rate} \quad & \text{(B.1e)}\\
\text{for } i &< j, & A_{ij}|\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, \boldsymbol{\eta} &\sim \mathrm{Pois}(\eta_{z_i z_j}) & \textit{link weight} \quad & \text{(B.1f)}
\end{aligned}$$

For the moment, we assume that $\boldsymbol{\mu}^{(1)}$ and $\boldsymbol{\mu}^{(2)}$ are *known and fixed*. With this specification, we can write up the joint distribution $p(\boldsymbol{A}, \boldsymbol{\eta}, \boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}|\boldsymbol{\psi})$, where for simplifying the notation we have introduced $\boldsymbol{\psi} \equiv (\alpha, \beta)^T$,

$$\begin{aligned}
&p(\boldsymbol{A}, \boldsymbol{\eta}, \boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}|\boldsymbol{\psi}) \\
&\quad = \Big(\prod_{ij} \mathrm{Pois}(A_{ij}|\eta_{z_i z_j})\Big)\Big(\prod_{lm} \mathrm{Gam}(\eta_{lm}|\alpha, \beta)\Big)\Big(\prod_{i} \mathrm{Mult}(z_i|\boldsymbol{\mu}^{(1)})\Big)\Big(\prod_{j} \mathrm{Mult}(z_j|\boldsymbol{\mu}^{(2)})\Big)
\end{aligned} \tag{B.2}$$

We start with the Gibbs sampling step for a single parameter $\eta_{l'm'}$ of the $\boldsymbol{\eta}$ matrix. By Bayes theorem,

$$p(\eta_{l'm'}|\boldsymbol{\eta}_{\setminus l'm'}, \boldsymbol{A}, \boldsymbol{\eta}, \boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, \boldsymbol{\psi}) = \frac{p(\boldsymbol{A}, \boldsymbol{\eta}, \boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}|\boldsymbol{\psi})}{\int p(\boldsymbol{A}, \boldsymbol{\eta}, \boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}|\boldsymbol{\psi}) d\eta_{l'm'}}. \tag{B.3}$$

In order to proceed, we reorder the terms of the joint distribution in order to separate out the parts that depend on $\eta_{l'm'}$. The parts that do not depend on $\eta_{l'm'}$, we group under a constant, C_η:

$$\begin{aligned}
&[p(\boldsymbol{A}, \boldsymbol{\eta}, \boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)} | \boldsymbol{\psi})]_{\eta_{l'm'}} \\
&\quad = C_\eta \cdot \eta_{l'm'}^{(\boldsymbol{Z}^{(1)} \boldsymbol{A} \boldsymbol{Z}^{(2)T})_{l'm'}} \cdot \exp(-\eta_{l'm'}) \cdot \eta_{l'm'}^{(\alpha-1)} \cdot \exp(-\beta \eta_{l'm'}) \qquad \text{(B.4)} \\
&\quad = C_\eta \cdot \eta_{l'm'}^{(\boldsymbol{Z}^{(1)} \boldsymbol{A} \boldsymbol{Z}^{(2)T})_{l'm'} + \alpha - 1} \cdot \exp(-\eta_{l'm'}(1+\beta)), \qquad \text{(B.5)}
\end{aligned}$$

where we have introduced $\boldsymbol{Z}^{(1)}$ being the $I \times L$ matrix obtained by stacking the cluster assignments $z_i^{(1)}$ as one-of-K encoded row-vectors, and $\boldsymbol{Z}^{(2)}$ likewise from the $z_j^{(2)}$ variables. In order to arrive at the above, we have expanded the Poisson and Gamma distributions of the likelihood and carefully isolated the parts depending on $\eta_{l'm'}$.

By noting that the above has the structure of the nominator in a Gamma distribution, in order for $p(\eta_{l'm'} | \boldsymbol{\eta}_{\setminus l'm'}, \boldsymbol{A}, \boldsymbol{\eta}, \boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, \boldsymbol{\psi})$ to sum to one, we know the normalizer must be exactly that of the Gamma distribution, hence we should sample,

$$p(\eta_{l'm'} | \boldsymbol{\eta}_{\setminus l'm'}, \boldsymbol{A}, \boldsymbol{\eta}, \boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, \boldsymbol{\psi}) = \mathrm{Gam}(\eta_{l'm'} | (\boldsymbol{Z}^{(1)} \boldsymbol{A} \boldsymbol{Z}^{(2)T})_{l'm'} + \alpha, 1 + \beta), \quad \text{(B.6)}$$

and in that way we avoid solving the integral of Eq. (B.3). This demonstrates how a conjugate prior for an exponential distribution likelihood, in this case the Poisson likelihood with Gamma prior, has a posterior of the same form as the prior, thus avoiding carrying out an intractable integral. For a general derivation see Bishop [11, p.117]. Note that for the above, we can sample the whole block of $\boldsymbol{\eta}$ parameters simultaneously, since the parameters do not depend on each other.

We now turn to how we obtain the sampling distributions of the marginals for $\boldsymbol{z}^{(1)}$ and $\boldsymbol{z}^{(2)}$. In the following we consider $z_i^{(1)}$ and write up Bayes theorem,

$$p(z_i^{(1)} = l | \boldsymbol{A}, \boldsymbol{\eta}, \boldsymbol{z}_{\setminus i}^{(1)}, \boldsymbol{z}^{(2)}, \boldsymbol{\psi}) = \frac{p(z_i^{(1)} = l, \boldsymbol{A}, \boldsymbol{\eta}, \boldsymbol{z}_{\setminus i}^{(1)}, \boldsymbol{z}^{(2)} | \boldsymbol{\psi})}{\sum_{l'} p(z_i^{(1)} = l', \boldsymbol{A}, \boldsymbol{\eta}, \boldsymbol{z}_{\setminus i}^{(1)}, \boldsymbol{z}^{(2)} | \boldsymbol{\psi})}. \qquad \text{(B.7)}$$

We then rewrite the joint probability in a functional form of $z_i^{(1)} = l$, which is simpler in the log-domain,

$$\begin{aligned}
\log [p(z_i^{(1)} = l | \boldsymbol{A}, \boldsymbol{\eta}, \boldsymbol{z}_{\setminus i}^{(1)}, \boldsymbol{z}^{(2)}, \boldsymbol{\psi})]_{z_i^{(1)} = l} & \\
&\propto \log(\mu_l^{(1)}) + \sum_j A_{ij} \log(\eta_{z_i z_j}) + \sum_j \eta_{z_i z_j} \\
&= \log(\mu_l^{(1)}) + \sum_m N_{im}^{\boldsymbol{A}} \log(\eta_{lm}) + N_m^{\boldsymbol{z}^{(2)}} \eta_{lm}, \qquad \text{(B.8)}
\end{aligned}$$

where we define $N_m^{\boldsymbol{z}^{(2)}} := \sum_j \delta_{m, z_j^{(2)}}$ and $N_{im}^{\boldsymbol{A}} := \sum_j A_{ij} \delta_{m, z_j^{(2)}}$. Since in the above each index i does not depend on any other index i', we can compute the whole block

of $\boldsymbol{z}^{(1)}$ simultaneously for each value of l, then obtaining an $I \times L$ matrix, which we exponentiate to revert from log-domain. Summing this matrix over the rows, we get the vector of normalization constants in Eq. (B.7) for all i and hence we can obtain samples from $p(z_i^{(1)} = l|\boldsymbol{A}, \boldsymbol{\eta}, \boldsymbol{z}_{\setminus i}, \boldsymbol{z}^{(2)}, \boldsymbol{\psi})$ all at once.

In the same manner, we get the conditional for $z_j^{(2)}$:

$$\begin{aligned}\log\left[p(z_j^{(2)} = m|\boldsymbol{A}, \boldsymbol{\eta}, \boldsymbol{z}^{(1)}, \boldsymbol{z}_{\setminus j}^{(2)}, \boldsymbol{\psi})\right]_{z_j^{(2)}=m} \\ \propto \log(\mu_m^{(2)}) + \sum_l N_{jl}^{\boldsymbol{A}} \log(\eta_{lm}) + N_l^{\boldsymbol{z}^{(1)}} \eta_{lm},\end{aligned} \tag{B.9}$$

where $N_l^{\boldsymbol{z}^{(1)}} := \sum_i \delta_{l,z_i^{(1)}}$ and $N_{jl}^{\boldsymbol{A}} := \sum_i A_{ij}\delta_{l,z_i^{(1)}}$.

The above then defines a blocked Gibbs algorithm that we can use for known L and M. If we wish to allow $L, M \to \infty$, i.e., a non-parametric model, then Ishwaran and James [44] presents techniques based on *stick-breaking priors*. For a non-parametric version of the procedure (B.1), we follow Ishwaran and James's algorithm for blocked Gibbs sampling [44, p. 168] with almost sure truncations [44, p. 165], which is a truncated stick-breaking construction of a finite dimensional Dirichlet process. This method still requires the maximum dimensions $L_{\max}$ and $M_{\max}$ to be specified, but for large enough values, the algorithm approximates the infinite process well. Using this approach, in each Gibbs sweep we can sample the same conditionals that we have introduced above, assuming $L = L_{\max}$ and $M = M_{\max}$ and at the end of each sweep, we then update $\boldsymbol{\mu}^{(1)}$ and $\boldsymbol{\mu}(2)$ according to a truncated stick-breaking construction, thereby approximating an infinite mixture and inferring still the *effective* number of clusters $L_{\text{eff}} \leq L_{\max}$ and $M_{\text{eff}} \leq M_{\max}$.

Analogous to the model for count networks, we can derive the blocked Gibbs sampler for the generative process of binary networks, which has the following specifications in place of Eq. (B.1e) and Eq. (B.1f),

$$\text{for } l \leq m, \qquad \eta_{lm}|\beta^+, \beta^- \sim \text{Beta}(\beta^+, \beta^-) \qquad \textit{link probability} \tag{B.10}$$

$$\text{for } i < j, \qquad A_{ij}|\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, \boldsymbol{\eta} \sim \text{Bern}(\eta_{z_i z_j}) \qquad \textit{link / no link} \tag{B.11}$$

The conditionals for a Gibbs sampler can be verified to become,

$$\begin{aligned}&p(\eta_{l'm'}|\boldsymbol{\eta}_{\setminus l'm'}, \boldsymbol{A}, \boldsymbol{\eta}, \boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, \boldsymbol{\psi}) \\ &\quad = \text{Beta}(\eta_{l'm'}|(\boldsymbol{Z}^{(1)}\boldsymbol{A}\boldsymbol{Z}^{(2)T})_{l'm'} + \beta^+, S_l^{\boldsymbol{z}^{(1)}} S_m^{\boldsymbol{z}^{(2)}} - (\boldsymbol{Z}^{(1)}\boldsymbol{A}\boldsymbol{Z}^{(2)T})_{l'm'} + \beta^-),\end{aligned} \tag{B.12}$$

where $S_l^{\boldsymbol{z}^{(1)}} := \sum_i \delta_{l,z_i^{(1)}}$ and $S_m^{\boldsymbol{z}^{(2)}} := \sum_j \delta_{m,z_j^{(2)}}$. And the required log-conditionals

for $\boldsymbol{z}^{(1)}$ and $\boldsymbol{z}^{(2)}$,

$$\begin{aligned}\log\left[p(z_i^{(1)} = l|\boldsymbol{A}, \boldsymbol{\eta}, \boldsymbol{z}_{\setminus i}^{(1)}, \boldsymbol{z}^{(2)}, \boldsymbol{\psi})\right]_{z_i^{(1)}=l} \\ \propto \log(\mu_l^{(1)}) + \sum_m N_{im}^{\boldsymbol{A}} \log(\eta_{lm}) + (S_m^{\boldsymbol{z}^{(2)}} - (\boldsymbol{A}\boldsymbol{Z}^{(2)})_{im}) \log(1-\eta_{lm})\end{aligned} \tag{B.13}$$

$$\begin{aligned}\log\left[p(z_j^{(2)} = m|\boldsymbol{A}, \boldsymbol{\eta}, \boldsymbol{z}^{(1)}, \boldsymbol{z}_{\setminus j}^{(2)}, \boldsymbol{\psi})\right]_{z_j^{(2)}=m} \\ \propto \log(\mu_m^{(2)}) + \sum_l N_{jl}^{\boldsymbol{A}} \log(\eta_{lm}) + (S_l^{\boldsymbol{z}^{(1)}} - (\boldsymbol{A}^T\boldsymbol{Z}^{(1)})_{jl}) \log(1-\eta_{lm}).\end{aligned} \tag{B.14}$$

Since the conditionals for $\boldsymbol{z}^{(1)}$ and $\boldsymbol{z}^{(2)}$ also separate for i and j respectively (in the log-domain), these can indeed also be sampled simultaneously. Furthermore, extending the model to sample $\mu_l^{(1)}, l = 1, \ldots, \infty$ and $\mu_m^{(2)}, m = 1, \ldots, \infty$ using the stick-breaking construction from Ishwaran and James [44], we infer the non-parametric model corresponding to the infinite relational model (IRM) [49].

APPENDIX C

Paper 1

Technical report

Predicting clicks in online display advertising with latent features and side-information

Bjarne Ørum Fruergaard
b.fruergaard@adform.com,
Adform Aps, Hovedvagtsgade 6 th., 1103 København K, Denmark

December 1, 2014

Abstract

We review a method for click-through rate prediction based on the work of Menon et al. [11], which combines collaborative filtering and matrix factorization with a side-information model and fuses the outputs to proper probabilities in $[0, 1]$. In addition we provide details, both for the modeling as well as the experimental part, that are not found elsewhere. We rigorously test the performance on several test data sets from consecutive days in a click-through rate prediction setup, in a manner which reflects a real-world pipeline. Our results confirm that performance can be increased using latent features, albeit the differences in the measures are small but significant.

Contents

1 Introduction

With the growing popularity of the Internet as a media, new technologies for targeting advertisements in the digital domain, a discipline generally referred to as *computational advertising*, have opened up to new business models for publishers and advertisers to finance their services and sell their products. Online advertising entails using banner ads as a means to attract user attention towards a certain brand or product. The clicks, known as *click-throughs*, take a user to a website specified by the *advertiser* and generates revenue for the page displaying the banner, which we call the *publisher*.

In *real-time bidding* (RTB) banner ads are determined and placed in real-time based on an auction initiated by the publisher between all potential advertisers, asking them to place a bid of what they are willing to pay for the current *impression* (displaying the ad), given information about the page, the user engaging the page, a description of the banner format and placement on the page. The advertiser with the highest bid wins the auction and their banner is displayed to the user. RTB thus requires advertisers, or more commonly, the *demand side platforms* (DSPs) acting on behalf of the advertisers, to be able to estimate the potential value of an impression, given the available information. A key measure for evaluating the potential values of impressions is the *click-through rate* (CTR), calculated as the ratio of the number of clicks over the total number of impressions in a specific context. What we are investigating in the present work, is a model for predicting CTRs, even in the face of contexts without any

previous clicks and/or very few impressions available, such that the empirical CTR can be unknown or very poorly estimated.

1.1 Dyadic prediction

We frame our main objective of estimating click-through rates for web banner advertisements in the general scope of a *dyadic prediction* task. Dyadic prediction concerns the task of predicting an outcome (or label) for a *dyad*, (i, j), whose members are uniquely identified by i and j, but which may include additional attributes of the dyad (i, j) being observed.

In this paper we are interested in predicting the binary labels being either *click* or *not click*, in general referred to as *click-through rate* prediction, given the pair of a *domain* and a web *banner* advertisement. In the following, we give a formal introduction of this problem.

We are given a transaction log of banner advertisements being shown to users. In the logs, various dimensions are recorded, including a *banner ID* and a *domain ID*, as well as a number of other attributes, which we shall elaborate more on later. For each record in the log, henceforth called a *view*, it is recorded whether the banner was clicked or it was displayed without any subsequent click (non-click). Let $i = 1, ..., M$ index the banner dimension and $j = 1, ..., N$ the domain dimension. We can then construct a matrix, $\boldsymbol{X}$, summarizing the records in the log in terms of empirical click-through rates, i.e., let the entries of the matrix be defined by

$$X_{ij} = \begin{cases} \frac{C_{ij}}{V_{ij}} & \text{if } V_{ij} > 0 \\ ? & \text{otherwise} \end{cases} \tag{1.1}$$

Here C_{ij} is the number of clicks and V_{ij} is the number of views involving dyad (i, j). Note that per definition, both clicks and non-clicks count as views, so we must always have $V_{ij} \geq C_{ij}$. The "?" denotes unobserved pairs, where there is no historical data in the log, hence for such dyads X_{ij} is undefined.

With this formulation, our click-through rate prediction task is to learn models estimating $\boldsymbol{X}$. Naturally, any such model should be able to predict the missing entries "?", as well as being able to *smoothen* predictions, such that the model does not get over-confident in situations with too few views. For instance, if $C_{ij} = 1$ and $V_{ij} = 3$, a CTR estimate of $X_{ij} = \frac{1}{3}$ is probably too extreme, as well as the case $C_{i'j'} = 0 \Rightarrow X_{i'j'} = 0$, where the natural assumption should rather be that not enough pairs (i', j') have yet been observed.

One possible approach to the above is where additional features about the entities i and j are known. This *side-information* can then be used as predictors in a supervised learning model, such as logistic regression. We refer to this approach as *feature-based*.

In the complete lack of side-information, one can instead borrow ideas from *collaborative filtering*. In collaborative filtering the classic setup, e.g., the Netflix movie rating problem [4], is where dyads are (user,item) pairs and each observed pair is labeled with a rating, for instance on the scale 1 to 5. The task is then to predict the ratings for unobserved pairs, which can then be used as a component in a recommender system. In our case we can identify a similar collaborative filtering task, but where instead of ratings we have binary outcomes and the dyads are (banner,domain) pairs. The assumption in collaborative filtering is that for a particular (banner,domain) pair, other row objects (other banners) as well as other column objects (other domains) contain predictive information. I.e., we are assuming that some information is *shared* between entities and we need to learn a model of this shared information.

In this work we investigate a model that fuses ideas from collaborative filtering via matrix factorization and a mapping to valid probabilities in $[0, 1]$, called a *latent feature log-linear model* (LFL) with a feature-based model for explicit features, that we refer to as a *side-information* model.

1.2 Related work

The model that we investigate in this work was introduced in [11] and builds on the latent feature log-linear model (LFL) from [12]. Our work can be seen as a supplement to [11], as we think this work is lacking in details, which we thus try and provide. Also, we offer different conclusions about the applicability of this model to a dataset of our own, but for the same application as [11]. [11] does not share any of their data so we can unfortunately not reproduce their results.

The modeling of click-through rates has been extensively investigated in the domain of search engine advertising, i.e., the sponsored advertisements that appear in web search engines as a result of user queries for relevant content retrieval. Many methods proposed in this domain are feature-based, e.g., [5, 7, 14] based on logistic regression. Other techniques are maximum likelihood based [3, 6], i.e., they operate directly with the empirically observed counts, which makes it a problem to predict in cold-start settings. Since in search engines, the user directly reveals an *intent* through his or her query, the features in most of these studies include somehow to predict click-through rates of pairs of (word,ad), which could indeed also be modeled using the LFL framework [12], but to our knowledge this has yet to be investigated.

In the setting that we are looking at, namely placement of banner ads, there is no direct query from the user, so modeling click-through rates cannot be based on (word,ad) pairs and have to be based on other often much weaker predictors of intent. Feature-based approaches are also popular in this setting, see e.g. [10]. Latent feature models are also not much explored in this area, hence a motivation for this work is to combine the best of combining latent and explicit features and share our findings.

Our focus in this work is on combining the LFL model [12] with a logistic regression model on the explicit features as in [11]. This combined model has the advantage that it in the face of weak explicit predictors, recommender effects from the latent features can kick in.

2 Response prediction

The model we apply for response prediction is based on the work in [11], a collaborative filtering technique based on matrix factorization, which is a special case of the latent feature log-linear model (LFL) [12] for dyadic prediction with binary labels. Menon et al. demonstrate that their model incorporating side-information, hierarchies and an EM-inspired iterative refinement procedure overcome many collaborative filtering challenges, such as sparsity and cold-start problems, and they show superior performance to models based purely on side-information and most notably the LMMH model [1]. In the following we introduce the *confidence-weighted* latent factor model from [11].

2.1 Confidence-weighted factorization

A binary classification problem of the probability of click given a dyadic observation (i, j) for page p_i and banner b_j, $p(click|(i, j))$, can be modeled with the *logistic function* and a single weight, ω_{ij}, per dyad. *I.e.*, $p^{LR}(click|\omega_{ij}) = \sigma(\omega_{ij})$. However, such a model is only capable of classifying dyads already observed in training data and cannot be applied to unseen combinations of pages and banners. Therefore we assume a factorization of ω_{ij} into the factors $\boldsymbol{\alpha}_i$ and $\boldsymbol{\beta}_j$ each representing latent feature vectors of the page and banner dimensions, respectively, such that $\omega_{ij} \approx \boldsymbol{\alpha}_i^T \boldsymbol{\beta}_j$. Henceforth, we will refer to this estimator as $p_{ij}^{MF} := p^{MF}(click|\boldsymbol{\alpha}_i, \boldsymbol{\beta}_j) = \sigma(\boldsymbol{\alpha}_i^T \boldsymbol{\beta}_j)$.

With data being $d = 1, ..., D$ observations of dyads, $x_d = (i, j)$, with binary labels, $y_d \in \{0, 1\}$, learning can be formulated as the regular logistic regression optimization problem:

$$\min_{\boldsymbol{A},\boldsymbol{B}} \quad -\sum_{d=1}^{D} y_d \log(p_{i_d j_d}^{MF}) + (1 - y_d) \log(1 - p_{i_d j_d}^{MF}), \tag{2.1}$$

i.e., a maximum-likelihood solution for Bernoulli-distributed output variables using the logistic function to non-linearly map continuous values to probabilities. With the latent variables $\boldsymbol{A}$ and $\boldsymbol{B}$ being indexed by (i, j), we can rewrite Eq. (2.1) to a *confidence-weighted factorization*:

$$\min_{\boldsymbol{A},\boldsymbol{B}} \quad -\sum_{(i,j)\in\mathcal{O}} C_{ij} \log(p_{ij}^{MF}) + (V_{ij} - C_{ij}) \log(1 - p_{ij}^{MF}), \tag{2.2}$$

where C_{ij} is the number of clicks ($y_d = 1$) involving dyad (i, j) and $(V_{ij} - C_{ij})$ the number of non-clicks ($y_d = 0$) involving dyad (i, j) in the training data. This reformulation can be a significant performance optimization, since the number of distinct dyads can be much smaller than the total number of observations. *E.g.*, in the case of click-through data, we can easily have many thousands of click and (particularly) non-click observations per dyad, hence the number of operations involved in the summation of Eq. (2.2) is significantly reduced compared to Eq. (2.1).

2.1.1 Regularization, learning and bias weights

Optimization of Eq. (2.2) is jointly non-convex in $\boldsymbol{A}$ and $\boldsymbol{B}$, but convex for $\boldsymbol{A}$ with $\boldsymbol{B}$ fixed, and vice versa. In practice that means we can only converge to a local minimum. Introducing regularization into the problem alleviates some non-convexity by excluding some local minima from the feasible set and additionally helps controlling overfitting. [11] suggests an ℓ_2 norm penalty, thereby effectively *smoothing* the latent factors:

$$\min_{\boldsymbol{A},\boldsymbol{B}} \quad \Omega_{\ell_2}(\boldsymbol{A}, \boldsymbol{B}) - \sum_{(i,j)\in\mathcal{O}} C_{ij} \log(p_{ij}^{MF}) + (V_{ij} - C_{ij}) \log(1 - p_{ij}^{MF}), \qquad (2.3)$$

where $\Omega_{\ell_2}(\boldsymbol{A}, \boldsymbol{B}) = \lambda(\sum_{i=1}^{I} ||\boldsymbol{\alpha}_i||_2^2 + \sum_{j=1}^{J} ||\boldsymbol{\beta}_j||_2^2)$. In this work we also try optimization with an ℓ_1 norm regularizer:

$$\min_{\boldsymbol{A},\boldsymbol{B}} \quad \Omega_{\ell_1}(\boldsymbol{A}, \boldsymbol{B}) - \sum_{(i,j)\in\mathcal{O}} C_{ij} \log(p_{ij}^{MF}) + (V_{ij} - C_{ij}) \log(1 - p_{ij}^{MF}), \qquad (2.4)$$

with $\Omega_{\ell_1}(\boldsymbol{A}, \boldsymbol{B}) = \lambda_\alpha \sum_{i=1}^{M} |\boldsymbol{\alpha}_i|_1 + \lambda_\beta \sum_{j=1}^{N} |\boldsymbol{\beta}_j|_1$, thereby promoting *sparse* latent features.

For the ℓ_2 regularized problem Eq. (2.3), a batch solver such as L-BFGS (see [13, Chapter 9]) can be invoked. For the ℓ_1 regularized problem Eq. (2.4), special care must be taken due to non-differentiability. The quasi-newton method OWL-QN [2] can be used instead in this setting. For really large problems, an on-line learning framework, such as *stochastic gradient descend* (SGD) is more scalable; again requiring special handling of the ℓ_1 regularizer; see [15] for details.

In general with classification problems with skewed class probabilities, *e.g.*, observing many more non-clicks than clicks, we can add bias terms to capture such baseline effects. We follow the suggestion from [12] and add separate bias terms for each row and column object, *i.e.*, in our case per-page and per-banner bias weights. Hence, without loss of generality, when we refer to $\boldsymbol{\alpha}_i$ and $\boldsymbol{\beta}_j$, we assume they have been appended $[\alpha_0, 1]^T$ and $[1, \beta_0]^T$, respectively, thereby catering for the biases. Furthermore, when we speak of a rank k latent feature model, we actually refer to a rank $k + 2$ model consisting of k latent features as well as the two bias dimensions.

2.2 Feature-based response prediction

A different approach to response prediction is a model based on explicit features available in each observation. In the case of click-through data, such information could for instance be attributes exposed by the browser (e.g., *browser* name and version, *OS*, *Screen* resolution, etc.), *time-of-day*, *day-of-week* as well as *user profiles* based on particular user's previous engagements with pages, banners, and with the ad server in general.

Again, we can use logistic regression to learn a model of binary labels: For $d = 1, ..., D$ observations we introduce *feature vectors*, $\boldsymbol{x}_d$, and model the probability of click given features with the logistic function, *i.e.*, $p_d^{LR} = p^{LR}(click|\boldsymbol{\omega}_{LR}) = \sigma(\boldsymbol{\omega}_{LR}^T \boldsymbol{x}_d)$. The optimization problem for learning the weights $\boldsymbol{\omega}_{LR}$ becomes

$$\min_{\boldsymbol{\omega}_{LR}} \quad \Omega_{\ell_1}(\boldsymbol{\omega}_{LR}) - \sum_{d=1}^{D} y_d \log(p_d^{LR}) + (1 - y_d) \log(1 - p_d^{LR}), \tag{2.5}$$

where $\Omega_{\ell_1}(\boldsymbol{\omega}_{LR}) = \lambda_{LR}|\boldsymbol{\omega}_{LR}|_1$ is added to control overfitting and produce sparse solutions. As discussed in Section 2.1.1, adding bias terms can account for skewed target distributions, and may be included in this type of model, *e.g.*, as a global intercept, by appending an all-one feature to all observations. Alternatively, if we want to mimic the per-page and per-banner biases of the latent factor model, we do so by including the page indices and banner indices encoded as one-of-M and one-of-N binary vectors, respectively, in the feature vectors.

2.3 Combining models

With dyadic response prediction as introduced in Section 2.1, the model can be extended to take into account *side-information* available to each dyadic observation. *I.e.*, introducing an order-3 tensor $\boldsymbol{X}$ with entries $\boldsymbol{x}_{ij} = \boldsymbol{X}_{ij:}$ being the feature vectors of side-information available to the (i, j) dyad, we follow [11] and model the confidence-weighted factorization with side-information as $p_{ij}^{SI} = p^{SI}(click|\boldsymbol{\alpha}_i, \boldsymbol{\beta}_j, \boldsymbol{\omega}_{SI}) = \sigma(\boldsymbol{\alpha}_i^T \boldsymbol{\beta}_j + \boldsymbol{\omega}_{SI}^T \boldsymbol{x}_{ij})$.

Learning such a model by jointly optimizing both $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, and $\boldsymbol{\omega}_{SI}$, is non-convex and may result in bad local minima [12]. To avoid such bad minima, [11, 12] suggest a simple heuristic; first learn a latent-feature model as the one detailed in Section 2.1, then train the side-information model as in Section 2.2, but given the log-odds $(\boldsymbol{\alpha}_i^T \boldsymbol{\beta}_j)$ from the latent-feature model as input. *I.e.*, p_{ij}^{SI} can be rewritten as

$$p_{ij}^{SI} = \sigma([\mathbf{1}; \boldsymbol{\omega}_{SI}]^T [\boldsymbol{\alpha}_i^T \boldsymbol{\beta}_j; \boldsymbol{x}_{ij}]), \tag{2.6}$$

hence, having first learned $\boldsymbol{\alpha}_i^T \boldsymbol{\beta}_j$ c.f. Section 2.1, $\boldsymbol{\omega}_{SI}$ can be learned by extending the input features with the log-odds of the latent-feature model and fixing

the corresponding weights to one. This training heuristic is a type of *residual fitting*, where the side-information model is learning from the differences between observed data and the predictions by the latent-feature model.

In practice we have found the above procedure to be insufficient for obtaining the best performance. Instead we need to *alternate* between fitting the latent features and fitting the side-information model, each while holding the predictions of the other model as fixed. This leaves how to train the latent feature model using the current side-information model prediction as fixed parameters open. Therefore we in the following show how this can be achieved.

For Eq. (2.3), which we will use as the working example, the observations are summarized for each unique dyad, (i, j), in terms of the click and non-click counts, regardless of the side-information in each of those observations. Therefore we now address the question: Given $\boldsymbol{\omega}_{LR}$ from Section 2.2, how do we obtain the quantities $\boldsymbol{\omega}_{SI}^T \boldsymbol{x}_{ij}$ in Eq. (2.6)?

Initially, we define the notation $\boldsymbol{x}_d^{(i,j)}$ as indexing the d^{th} explicit feature vector involving the dyad (i, j). Hence, for dyad (i, j) there are V_{ij} (potentially) different feature vectors $\boldsymbol{x}_d^{(i,j)}$, $d = 1, ...V_{ij}$, involved. Assuming a model learned on the explicit features alone according to p_d^{LR} from Section 2.2, the overall predicted click-through rate for the observations involving dyad (i, j) becomes

$$p_{ij}^{LR} := \frac{1}{V_{ij}} \sum_{d=1}^{V_{ij}} \sigma(\boldsymbol{\omega}_{LR}^T \boldsymbol{x}_d^{(i,j)}), \tag{2.7}$$

which is obvious from the fact that the sum calculates the predicted number of clicks and V_{ij} is the empirical number of observations. *I.e.*, Eq. (2.7) is just the average predicted click-through rate taken over the observations involving dyad (i, j). Using this result we can now make sure the combined model yields p_{ij}^{LR} when either $\boldsymbol{\alpha}_i = \mathbf{0}$ or $\boldsymbol{\beta}_j = \mathbf{0}$ (or both) by fixing the term $\boldsymbol{\omega}_{SI} \boldsymbol{x}_{ij}$ according to the log-odds of p_{ij}^{LR}. Hence,

$$\boldsymbol{\omega}_{SI}^T \boldsymbol{x}_{ij} = \log(p_{ij}^{LR}) - \log(1 - p_{ij}^{LR}) \tag{2.8}$$

should be used as fixed inputs while learning the latent-factor model and thus accounts for the predictions of the feature-based model the same way as bias terms account for baseline effects.

3 Data and experiments

We will run experiments on datasets extracted from ad transaction logs in Adform, an international online advertising technology provider. Due to the sequential nature of these data, we will report results from training a model on 7 consecutive days and then testing on the 8^{th}. For measuring performance

we report area under the ROC curve (AUC) scores as well as the logistic loss, evaluated on the held-out data (i.e., the last day). We evaluate different instantiations of the model over a period of in total 23 test days each using the previous 7 days for training and therefore can also report on the consistency of the results.

The data consists of observations labeled either click or not click and in each observation the domain and the banner id are always known. The additional features, that are features for the side-information, include various categorical variables all encoded as one-of-K. These include the web browsers *UserAgent* string (a weak fingerprint of a user), an indicator vector of the top-50k (for a single country) websites the user has visited (*URLs visited*) the past 30 days, the full URL of the site being visited (top-50k indicator vector, per country), a binned frequency of the times the user has clicked before (never, low, mid, high), as well as cross-features of the above mentioned and each banner id, thereby tailoring coefficients for each ad. The resulting number of features (P) is between 500k-600k and the number of positive observations is around 250k (N_1), i.e., the problem is overcomplete in features and thus ℓ_1 on the side-information model is added as a means of feature selection. The negative class N_0 is down-sampled by a factor of 100 bringing it down to around 1.5M-2.5M. The resulting model can be corrected in the intercept [8]. In our experience down-sampling the negative class this drastically and calibrating the model by intercept correction does not impact downstream performance.

We train different variations of the models to investigate in particular the usefulness of latent features in addition to a model using only explicit features. The different models are:

LR Logistic regression on the side-information alone. The corresponding regularization strength we call λ_{LR}.

LFL$_0$ The latent feature log-linear model using only the bias features, i.e., corresponding to a logistic regression for the two indicator features for domain and banner, respectively. The corresponding regularization weights we refer to as λ_{α_0} and λ_{β_0}, but as we describe later, in practice we use the same weight, λ_0, for both.

LFL$_K$ The latent feature log-linear model with $K >= 0$ including bias features. The corresponding regularization weights we call λ_α and λ_β, which we also find in practice can be set to be equal, and for this introduce the weight $\lambda_{\alpha\beta}$.

LR+LFL$_K$ The combined model with $K \geq 0$ for the LFL model combined with the side-information model.

3.1 Tuning hyper-parameters

The combined model with both latent features and side-information we dub LR+LFL$_K$. This model has up to 5 (!) hyper-parameters that need tuning by cross-validation: $(\lambda_{LR}, \lambda_{\alpha_0}, \lambda_{\beta_0}, \lambda_\alpha, \lambda_\beta)$. [12] does not report whether they use individual λ_α, λ_β, λ_{α_0}, and λ_{β_0} weights, but we consider this highly infeasible. What we have found to be most effective, is to use the same weight $\lambda_{\alpha\beta}$ for the latent dimensions as well as a shared bias weight λ_0, which narrows the search space down to three hyper-parameters that must be tuned.

Tuning three hyper-parameters is still a cumbersome task, in particular for large datasets, where an exhaustive search for a reasonable grid of three parameters becomes too time consuming. Instead we have had success using the following heuristic strategy for tuning of these parameters:

1. First run experiments for the logistic regression model alone and find a suitable regularization weight λ_{LR}.
2. Run experiments for the LFL$_0$ model (i.e., bias weights only) and find a suitable λ_0.
3. Run experiments for a number of LFL$_K$ models with $K > 0$, with bias weights regularized by λ_0 fixed from (2), and find a suitable $\lambda_{\alpha\beta}$.
4. Finally, train the combined LFL+LR$_K$ model with different $K \geq 0$ and λ_0 fixed, but varying $\lambda_{\alpha\beta}$ as well as λ_{LR} both in the *neighborhood* of the values found in (1) and (3). If the results indicate performance could be improved in any direction away from that region, we run more experiments with the hyper-parameters set in that direction.

To verify the validity of this approach, we have run experiments with the hyper-parameters set to their optimal settings as per the above procedure, and varying one at the time, including separate weights for the latent features and biases. In this way we do not find any increase in the performance along any single direction in the space of hyper-parameters.

4 Results and discussion

4.1 Validation set results and initialization

We use the first 8 days, i.e., train on 7, test on the 8$^{\text{th}}$, to find a reasonable range of hyper-parameters that we will test over the entire period. I.e., we use the first test day of a total of 23 days (30 days worth of data, where the first 7 are only used for training) as our *validation set*. At the same we initialize models with different parameters, that we use for *warm starting* the training on subsequent data. In the following we provide our results where we are testing performance on a single day, thereby gaining insights into both hyper-parameter

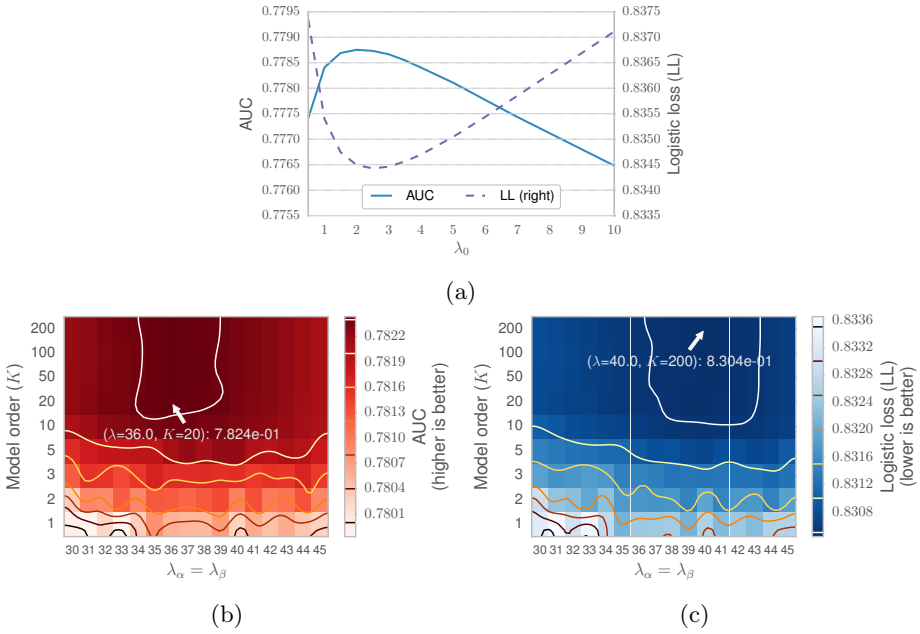


Figure 1: Results using ℓ_2 regularization for LFL_K modeling on a single day with varying regularization strengths and model orders. (a) A sweep in over the LFL_0 regularization strength λ_0 in the vicinity of the optimum. (b) Using $\lambda_0 = 3.0$ the AUC is plotted as intensities in a grid with varying model orders and the shared regularization strength $\lambda_{\alpha\beta}$ for the latent dimensions. The annotations marks the optimum. (c) Same as (b), except for logistic loss, i.e., the lower the better.

values, model order (K) and regularization type (ℓ_1 and ℓ_2) for the latent features.

In Fig. 1 we show the results using ℓ_2 regularization (see Eq. (2.3)) and varying λ_0 with an LFL_0 model (a) and $\lambda_{\alpha\beta}$ in (b-c) with $\lambda_0 = 3.0$ fixed, different model orders and no side-information model. Results are not shown for experiments where λ_0 and $\lambda_{\alpha\beta}$ were varied in larger grids, i.e., these plots focus of where the performances peak. What we also learn from these plots (b-c), is that higher model orders are advantageous, but that this increase levels off from between $K = 5$ to $K = 20$. This is in contrast to [11] reporting $K \geq 200$ being advantageous. We have also run experiments not shown here with $K = 100$ and $K = 200$ and seen no further increase, and if anything at all, a slight decrease in performance.

The same experiments using ℓ_1 regularization are summarized in Fig. 2. Both the experiments for the bias regularization λ_0 (a) as well as those for the latent factors (b-c) do not show as good performances as in the case of ℓ_2 regular-

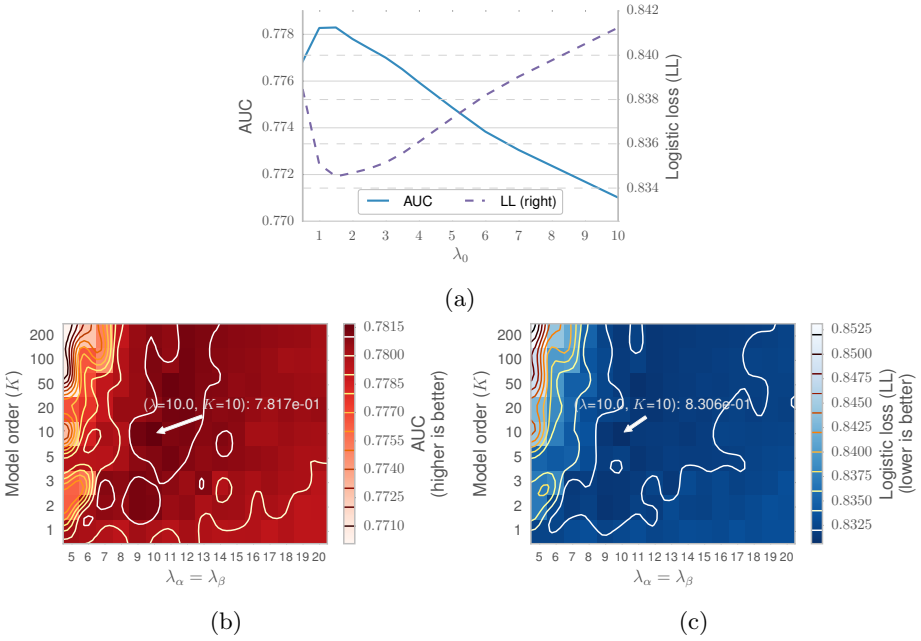


Figure 2: Results using ℓ_1 regularization for LFL_K modeling on a single day with varying regularization strengths and model orders. Also see the caption for Fig. 1.

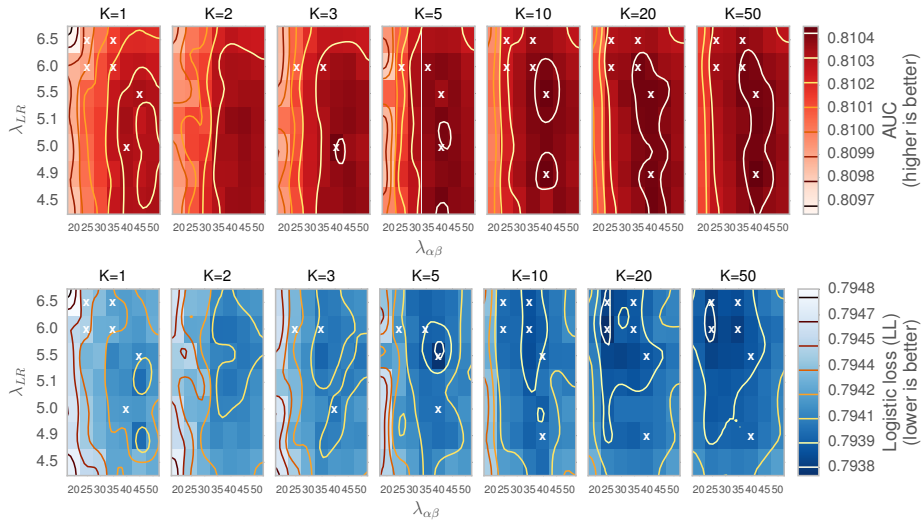


Figure 3: Results using the ℓ_2 regularized version for LR+LFL$_K$ on the first day and varying regularization strengths as well as model orders. $\lambda_0 = 3.0$ remains fixed. In the top we show AUC intensities and in the bottom logistic losses. Little x's are used to mark those specific configurations that we run experiments with across all the test days.

ization and the advantage of adding latent dimensions is harder to distinguish. Furthermore the regions of interest seem more concentrated, i.e., the optima are more peaked. This leads us to the conclusion, that smoothness in the latent dimensions (ℓ_2) is preferable to sparsity (ℓ_1) and thus we do not report further results using ℓ_1 regularization.

In Fig. 3 we show experiments with varying λ_{LR}, $\lambda_{\alpha\beta}$ as well as different models orders K for combined LR+LFL$_K$ models. We confirm a trend towards better performance using higher K, but again saturating beyond $K = 5$. We further notice that peak performances in terms of AUC do not necessarily agree completely with those for LL. There may be other explanations as to why that is, but we believe this is a consequence of the LL being sensitive to probabilities being improperly calibrated, while the AUC is not. Inspection of the different models seem to confirm this; where the models perform better in terms of logistic loss, the predicted click-through rate for the test set is (slightly) closer to the empirical, than for those models which maximize AUC. We expect that a post-calibration of the models beyond just an intercept correction could be beneficial for the reported logistic losses, but also note that this would not change the AUCs.

As mentioned in Section 2.3, we find that alternating between fitting the latent model and the side-information model is necessary. For the experiments Fig. 3, we have alternated 7 times which we have confirmed in practice ensures the

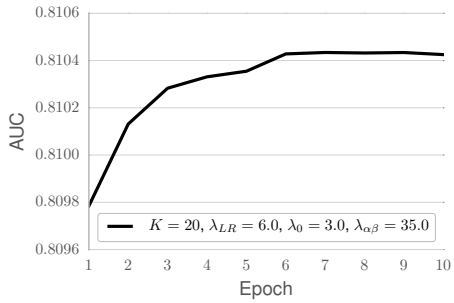


Figure 4: An example of a run of the LR+LFL$_K$ model with alternating updates to the latent and the side-information coefficients, which illustrates the generic level-off in performance (here AUC), as we run more epochs.

performance has leveled off. An example supporting this claim is shown in Fig. 4 and serves to illustrate the general observation we make in all our experiments.

4.2 Results on 22 consecutive days

With just a subset of LR+LFL$_K$ models hand-picked (marked by little x's in Fig. 3) from the experiments on the first test day, we run experiments on a daily basis while initializing with the models from the previous day. This sequential learning process reflects how modeling would also be run in production at Adform and by warm starting the models in the previous days' coefficients, we do not expect that running multiple epochs of alternated fitting is required, i.e., this only needs to be done once for initialization.

In the following, the AUCs and logistic losses we report are daily averages of the performances for each banner. As opposed to the performances over an entire test data set that we have reported up until now, making daily averages per banner prevents the performance numbers from being entirely dominated by a single or a few banners, and instead assigns per-banner performances equal weights.

Reporting performances based on slices of data per banner further allows analysis of under which circumstances the latent feature models add a statistical significant improvements. In Fig. 5 we show the difference in AUC banner averages per day in the total of 22 days we use for testing. The upper shows the performances for all the banners with 1 or more clicks in each test set (day), while the lower is averaged daily performances for only the banners with 10 or more clicks. It is apparent from these two figures, that AUC scores based on very few clicks add significant variance to the daily averages and the difference between the model orders is hard to spot. We also note, that since we cannot evaluate AUCs score for banners without clicks in the test set, these are ignored entirely. For logistic loss, however, we can still report a performance for banners

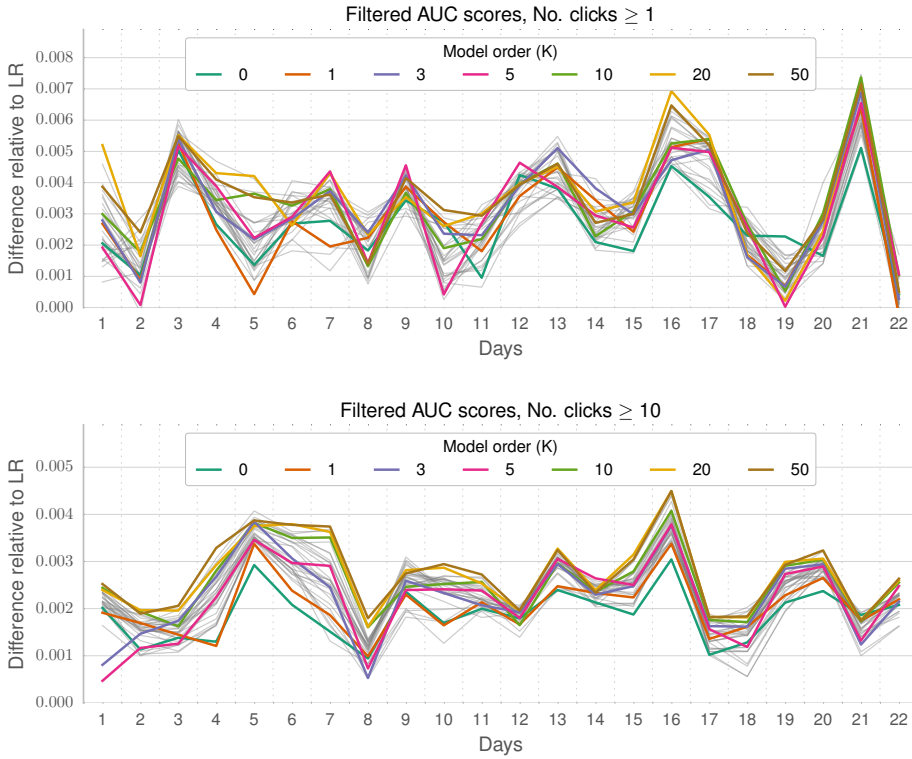


Figure 5: Daily average AUC differences (i.e., increase) for LR+LFL$_K$ models using the optimal settings for different model-orders (colored lines) relative to the side-information model alone. Shaded, gray lines in the background trace all of the different configurations tested. In the above the averages include every banner with 1 or more clicks on a day (between 900-1000 banners qualify each day), while in the lower all the banners on a particular day with less than 10 clicks is filtered from the averages (between 400-500 banners qualify each day), hence decreasing the variance in the measures.

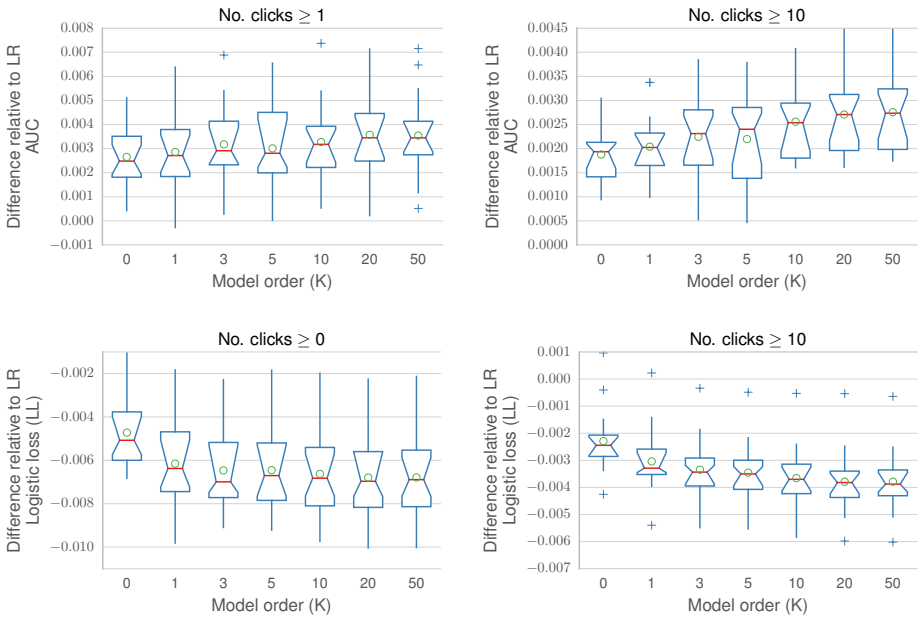


Figure 6: **Top row**: Box plots of the relative AUC differences corresponding to each of the models also displayed in Fig. 5. **Bottom row**: Box plots also for the relative logistic loss differences with the only difference being that in the left-most box plot, the losses for *all* banners each day are included in the averaging (1100-1200 daily). **Legend**: Boxes show 1st and 3rd quartiles, whiskers are 1.5 IQR (interquartile range), medians are red lines and green circles mark the means. Outliers are marked with pluses. The notches are 5000 bootstrap sample estimates of the 5%-95% confidence interval for the medians.

without clicks in the test. The LR model used as the reference (0.0) in Fig. 5 uses $\lambda_{LR} = 4.0$, which we found as optimal over the entire 22 day period testing a grid from 2.0 to 7.0 in increments of 0.5.

In order to further quantify and investigate the impact different model orders has on performance, we summarize in Fig. 6 the relative differences over the 22 test days in box plots. Again, we show performances relative to the side-information model and for different inclusion criteria, based on number of clicks in the test sets. In all cases, we see an increase in performance, as the model order is increased, and this increase levels off from $K = 10$ to $K = 50$. The notches on boxes are 5k sample bootstraps of the medians, hence based on these we can say something about the statistical significance of these results. I.e., non-overlapping notches correspond to $p < 0.05$ for a two-tailed null-hypothesis test. First of all, all model orders, including $K = 0$, improve performances compared to the side-information model alone.

For both the AUCs and the logistic losses we see wide confidence intervals on

the medians, when banners with very few clicks (< 10) per day are included. We still observe an increase in performance as the model order increases, but only in the case of logistic loss do the model orders $K = 20$ and $K = 50$ barely clear overlapping with the notches of $K = 0$.

In the case of including only banners with more than 10 clicks in the summary statistics, the confidence intervals of the medians shrink, in particularly in the case of logistic loss. However, the relative gains (means and medians) are also slightly lower. I.e., there is a *trend*, albeit barely statistically significant, that there are higher gains among the banners with few clicks in the test sets, than for those with more. Apart from this, there is now also statistically significant differences between the medians for the higher model orders and $K = 0$; in the case of AUC this includes $K = 20$ and $K = 50$, and in the case of logistic loss, $K \geq 3$ are statistically better.

It is worth noting that, regardless of the slice based on number of clicks in the test sets, the results agree that using the LR+LFL$_K$ model yields higher performance than the LR model alone.

For the results in Fig. 6, while we find evidence that supports that latent features improves click-through prediction, the question remains *how much* this improves real-world performances. Indeed the increments which the latent features introduce in the two measures we report here seem very small. When measuring AUC scores, in particular, we are however not the first to report small, but significant improvements on the third decimal. As McMahan [9] (on web search ads) puts it:

> *The improvement is more significant than it first appears. A simple model with only features based on where the ads were shown achieves an AUC of nearly 0.80, and the inherent uncertainty in the clicks means that even predicting perfect probabilities would produce an AUC significantly less than 1.0, perhaps 0.85.*[9, p.532]

Our data as well as our experiences in web banner ads support this statement, and we often also identify new features or model changes with these low levels of improvement, but which however remain consistent.

Another possibility as an alternative to *off-line* measures on held-out data, such AUC and logistic loss, is *live* A/B testing. Yet, before taking new features, models or technology into production, a prerequisite to us at least, is to demonstrate consistent off-line data performance improvements. For the present work, we have not had the opportunity to test it live.

5 Conclusion

In this work we have reviewed a method for click-through rate prediction which combines collaborative filtering and matrix factorization with a side-information

model and fuses the outputs to proper probabilities in $[0, 1]$. We have provided details about this particular setup that are not found elsewhere and shared results from numerous experiments highlighting both the strengths and the weaknesses of the approach.

We test the model on multiple consecutive days of click-through data from Adform ad transaction logs in a manner which reflects a real-world pipeline and show that predictive performance can be increased using higher-order latent dimensions. We do see a level-off in the performances for $\approx K \geq 20$, whereas $K \geq 200$ was suggested in another work [11], but may be due to differences in the data sets; in particular how many side-information features are available and used.

Our numerous experiments detail a very involved phase for finding proper regions for the various hyper-parameters of the combined model. This is particularly complicated, since the latent feature model and the side-information model need to be trained in several alternating steps, for each combination of hyper-parameters. This we think is one of the most severe weaknesses of this modeling approach. We circumvent some of the complexity of finding good hyper-parameters by using shared regularization strengths for both entities of the latent model and demonstrate, that in a sequential learning pipeline, it is only for initialization of the model, i.e., on the first training set, that we need multiple alternating steps.

For future studies, it would be particularly useful if the hyper-parameters could instead be inferred from data. Yet, as we also show in our results, the objective differences (i.e., the *evidence*) that separate good models from the bad, are small, hence we expect any technique, such as Type II maximum likelihood, would be struggling to properly navigate such a landscape.

References

[1] Agarwal, D., Agrawal, R., Khanna, R., and Kota, N. Estimating rates of rare events with multiple hierarchies through scalable log-linear models. *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '10*, page 213, 2010.

[2] Andrew, G. and Gao, J. Scalable training of L1-regularized log-linear models. *Proceedings of the 24th international conference on Machine learning*, pages 33–40, 2007.

[3] Ashkan, A., Clarke, C. L. a., Agichtein, E., and Guo, Q. Estimating Ad Clickthrough Rate through Query Intent Analysis. *2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, pages 222–229, 2009.

[4] Bennett, J. and Lanning, S. The netflix prize. *In KDD Cup and Workshop in conjunction with KDD*, 2007.

[5] Chakrabarti, D., Agarwal, D., and Josifovski, V. Contextual advertising by combining relevance with click feedback. *Proceedings of the 17th international conference on World Wide Web*, pages 417–426, 2008.

[6] Dembczynski, K., Kotlowski, W., and Weiss, D. Predicting ads' clickthrough rate with decision rules. *Workshop on Targeting and Ranking in Online Advertising*, 2008.

[7] Graepel, T., Borchert, T., Herbrich, R., and Com, R. M. Web-Scale Bayesian Click-Through Rate Prediction for Sponsored Search Advertising in Microsofts Bing Search Engine. *Search*, (April 2009), 2010.

[8] King, G. and Zeng, L. Explaining Rare Events in International Relations. *International Organization*, 55(3):693–715, September 2001.

[9] McMahan, H. Follow-the-regularized-leader and mirror descent: Equivalence theorems and l1 regularization. *International Conference on Artificial Intelligence and Statistics*, 2011.

[10] McMahan, H., Holt, G., and Sculley, D. Ad click prediction: a view from the trenches. *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2013.

[11] Menon, A., Chitrapura, K., and Garg, S. Response prediction using collaborative filtering with hierarchies and side-information. *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining KDD*, 2011.

[12] Menon, A. A. K. and Elkan, C. A log-linear model with latent features for dyadic prediction. *2010 IEEE International Conference on Data Mining*, pages 364–373, December 2010.

[13] Nocedal, J. and Wright, S. J. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer-Verlag, New York, 1999.

[14] Richardson, M., Dominowska, E., and Ragno, R. Predicting clicks: estimating the click-through rate for new ads. *Proceedings of the 16th international conference on World Wide Web*, pages 521–530, 2007.

[15] Tsuruoka, Y., Tsujii, J., and Ananiadou, S. Stochastic gradient descent training for L1-regularized log-linear models with cumulative penalty. *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1 - ACL-IJCNLP '09*, 1:477, 2009.

APPENDIX D

Paper 2

Dimensionality reduction for click-through rate prediction: Dense versus sparse representation

Bjarne Ørum Fruergaard, Toke Jansen Hansen
Adform ApS
Hovedvagtsgade 6
DK-1103 Copenhagen K, Denmark
{b.fruergaard,t.hansen}@adform.com

Lars Kai Hansen
Technical University of Denmark
Anker Engelunds Vej 1,
DK-2800 Kgs. Lyngby, Denmark
lkai@dtu.dk

Abstract

In online advertising, display ads are increasingly being placed based on real-time auctions where the advertiser who wins gets to serve the ad. This is called real-time bidding (RTB). In RTB, auctions have very tight time constraints on the order of 100ms. Therefore mechanisms for bidding intelligently such as click-through rate prediction need to be sufficiently fast. In this work, we propose to use dimensionality reduction of the user-website interaction graph in order to produce simplified features of users and websites that can be used as predictors of click-through rate. We demonstrate that the Infinite Relational Model (IRM) as a dimensionality reduction offers comparable predictive performance to conventional dimensionality reduction schemes, while achieving the most economical usage of features and fastest computations at run-time. For applications such as real-time bidding, where fast database I/O and few computations are key to success, we thus recommend using IRM based features as predictors to exploit the recommender effects from bipartite graphs.

1 Introduction

In advertising, one is interested in segmenting people and targeting ads based on segments [1]. With the rapid growth of the Web as a publishing platform, new advertising technologies have evolved, offering greater reach and new possibilities for targeted advertising. One such innovation is real-time bidding (RTB), where upon a user's request for a specific URL, an online real-time auction is started amongst numerous participants, competing to serve their advertisement. The participants are allotted a limited time on the order of 100ms to query their data sources and come up with a bid, and the winner gets to display their advertisement. Thus if the computational complexity can be reduced, more complex decision processes can be invoked. In this work, we evaluate how dimensionality reduction can be used to simplify predictors of click-through rate.

We focus on three techniques for dimensionality reduction of the large bipartite graph of user-website interactions, namely Singular Value Decomposition (SVD) [2], Non-negative Matrix Factorization (NMF) [3], and the Infinite Relational Model (IRM) [4]. We are interested in how the different levels of sparsity of the output features imposed by each of the models affect the performance in a click-through rate prediction task. In the RTB setup, where low latency and high throughput are both of crucial importance, database queries need to require as little I/O as possible, and computing model predictions need to involve as few operations as possible. Therefore a good idea is to "compress" very high-cardinality features using dimensionality reduction techniques and at the same time potentially benefit from recommender effects [5]. This presents a trade-off between how much to compress in order to speed up I/O and calculations versus retaining, or exceeding, the performance of a high cardinality feature.

By investigating the SVD, NMF, and the IRM, we essentially vary the compression of a high-cardinality feature (user-website engagements). The SVD produces dense singular vectors, thus requiring the most I/O as well as computation. The NMF is known to produce sparse components [3], meaning that zeros need not be stored, retrieved nor used in computations, and thus requires less I/O and computation. The IRM offers the most sparse representation, in that it produces hard cluster assignments, hence I/O and computation are reduced to a single weight per mode.

We present results that use either of the dimensionality reduction techniques' outputs as predictors for a click-through rate prediction task. Our experiments show that a compact representation based on the NMF outperforms the other two options. If one however wants to use as little I/O and as simple computations as possible, the very compact representation from the IRM model offers an interesting alternative. While incurring a limited loss of lift relative to the NMF, the IRM based predictors yield the fastest training speed of the downstream logistic regression classifier and also results in the most economical usage of features and fastest possible computations at run-time. The IRM further has the advantage that it alleviates the need for model order selection, which is required in NMF. While the dense features produced by SVD also find usage in terms of predictive performance, the dense features inhibit the logistic regression training time, and if low database I/O as well as fast computation of predictions is a priority, the SVD will not be of great use.

A key enabling factor in running the IRM with the data we present in this work, is a sampler written for the graphics processing unit (GPU) [6], without which learning of the IRM model would not be feasible, at least not on a day-by-day schedule. To demonstrate the feasibility of the IRM as a large-scale sparse dimensionality reduction, we run final tests on a full-scale click-through rate data set and compare the performances with not using any dimensionality reductions.

1.1 Related work

Within the area of online advertising, computational targeting techniques are often faced with the challenge of very few observations per feature, particularly of the positive label (i.e., click, action, buy). A common approach to alleviate such label sparsity is to use collaborative filtering type algorithms, where one allow similar objects to "borrow" training data and thus constrain the related objects to have similar predicted behaviour. Studies hereof are common for sponsored search advertising where the objects of interest are query-ad pairs [5, 7], but the problem is similar to that of user-website pairs that we study. To our knowledge we are the first to report on the usage of the IRM co-clustering of user-website pairs and the results should be applicable for query-add click-through rate prediction as well.

By representing users in a compressed or latent space based on the user-website graph, we are essentially building profiles of users based on their behaviour and using those profiles for targeted advertising. This approach is well studied with many other types of profiles based on various types of information: For using explicit features available for predicting click-through rates, [8] is a good resource: Latent factor models have been proposed to model click-through rates in online advertising, see e.g. [9]: For examples of using dimensionality reduction techniques in the construction of click-through rate models, such as the NMF, see [10]. We believe our contribution to have applications in many such setups, either as an additional predictor or for incorporation as *a priori* information (priors, constraints, etc.) which can help with identifiability of the models.

We regard the problem of predicting click-through rates as a supervised learning task, i.e., given historical observations with features (or predictors) available about the user, webpage, and ad, along with the labels of actions (in our case click (1) or not-click (0)), the task is to learn a classifier for predicting unseen observations, given the features. This is the approach taken also by e.g., [8]. As in [8], we build a probabilistic model based on logistic regression for predicting click-through rates. What we add, is additional features based on dimensionality reduction, as well as a sparsity inducing constraint based on the L_1-norm.

2 Methods

We are interested in estimation of features which can improve click-through rate predictions. In this work, we focus on introducing features from different dimensionality reduction techniques based on a bipartite graph of users and websites (URLs), and using them in a simple probabilistic model

for click-through rate prediction, namely logistic regression. In the following, we introduce the dimensionality reduction techniques which we evaluate.

2.1 Dimensionality reduction techniques

2.1.1 Singular value decomposition

The singular value decomposition (SVD) of a rank R matrix $\boldsymbol{X}$ is given as the factorization $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top = \sum_{i=1}^{R} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^\top$, where $\boldsymbol{U}$ and $\boldsymbol{V}$ are unitary matrices $\boldsymbol{U}^\top\boldsymbol{U} = \boldsymbol{V}^\top\boldsymbol{V} = \boldsymbol{I}$ and hold the left and right singular vectors of $\boldsymbol{X}$, respectively. The diagonal matrix $\boldsymbol{\Sigma}$ contains the singular values of $\boldsymbol{X}$. By selecting only the K largest singular values of $\boldsymbol{\Sigma}$, i.e., truncating all other singular values to zero, one obtains the approximation $\tilde{\boldsymbol{X}} = \boldsymbol{U}\tilde{\boldsymbol{\Sigma}}\boldsymbol{V}^\top = \sum_{i=1}^{K} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^\top$, which is the rank K optimal solution to $\arg\min ||\boldsymbol{X} - \tilde{\boldsymbol{X}}||_2^2$. This truncation corresponds to disregarding the $R - K$ dimensions with the least variances of the bases $\boldsymbol{U}$ and $\boldsymbol{V}^\top$ as noise.

2.1.2 Non-negative matrix factorization

Non-negative matrix factorization (NMF) received its name as well as its popularity in [3]. NMF is a matrix factorization comparable to SVD, the crucial difference being that NMF decomposes into non-negative factors and impose no orthogonality constraints. Given a non-negative input matrix $\boldsymbol{X}$ with dimensions $M \times N$, NMF approximates the decomposition $\boldsymbol{X} \approx \boldsymbol{W}\boldsymbol{H}$, where $\boldsymbol{W}$ is an $M \times K$ non-negative matrix, $\boldsymbol{H}$ a $K \times N$ non-negative matrix, and K is the number of components. By selecting $K << min(M, N)$ one approximates the decomposition of $\boldsymbol{X}^{(M\times N)} = \boldsymbol{W}^{(M\times K)}\boldsymbol{H}^{(K\times N)} + \boldsymbol{E}^{(M\times N)}$, thereby disregarding some residual (unconstrained) matrix $\boldsymbol{E}$ as noise.

NMF has achieved good empirical results as an unsupervised learning technique within many applications, e.g., for document clustering [11, 12, 13], visual coding [3], and bioinformatics [14]. For NMF applications for computational advertising, see also [10].

2.1.3 Infinite relational model

The Infinite Relational Model (IRM) has been proposed as a Bayesian generative model for graphs. Generative models can provide accurate predictions and through inference of relevant latent variables they can inform the user about mesoscale structure. The IRM model can be cast as co-clustering approach for bipartite networks where the nodes of each mode are grouped simultaneously. A benefit of the IRM model over existing co-clustering approaches is that the model explicitly exploit the statistical properties of binary graphs and allows the number of components of each mode to be inferred from the data.

The generative process for the the Relational Model [4, 15, 16] is given by:
$\diamond$ Sample the row cluster probabilities, i.e., $\boldsymbol{\mu}^{(1)} \sim Dirichlet(\alpha^{(1)}/K^{(1)}\boldsymbol{e}^{(1)})$.
$\diamond$ Sample row cluster assignments, i.e., $m = 1, \ldots, M$ $\boldsymbol{z}_m^{(1)} \sim Discrete(\boldsymbol{\mu}^{(1)})$.
$\diamond$ Sample the column cluster probabilities, i.e., $\boldsymbol{\mu}^{(2)} \sim Dirichlet(\alpha^{(2)}/K^{(2)}\boldsymbol{e}^{(2)})$.
$\diamond$ Sample column cluster assignments, i.e., $n = 1, \ldots, N$ $\boldsymbol{z}_n^{(2)} \sim Discrete(\boldsymbol{\mu}^{(2)})$.
$\diamond$ Sample between cluster relations, i.e., $i = 1, \ldots, I$ and $j = 1, \ldots, J$ $\eta_{ij} \sim Beta(\beta^+, \beta^-)$.
$\diamond$ Generate links, i.e., $m = 1, \ldots, M$ and $n = 1, \ldots, N$ $X_{nm} \sim Bernoulli(\boldsymbol{z}_n^{(1)\top}\boldsymbol{\eta}\boldsymbol{z}_m^{(2)})$.
Where $K^{(1)}$ and $K^{(2)}$ denote the number of row and column clusters respectively whereas $\boldsymbol{e}^{(1)}$ and $\boldsymbol{e}^{(2)}$ are vectors of ones with size $K^{(1)}$ and $K^{(2)}$. The limits $K^{(1)} \to \infty$ and $K^{(2)} \to \infty$ lead to the Infinite Relational Model (IRM) which has an analytic solution given by the Chinese Restaurant Process (CRP) [15, 4, 17].

Rather than collapsing the parameters of the model, we apply blocked sampling that allows for parallel GPU computation [6]. Moreover, the CRP is approximated by the truncated stick breaking construction (TSB), and the truncation error becomes insignificant when the model is estimated for large values of $K^{(1)}$ and $K^{(2)}$, see also [18].

2.2 Supervised learning using logistic regression

For learning a model capable of predicting click-through rates trained on historical data, we employ logistic regression with sparsity constraints; for further details see for instance [19, 20]. Given data consisting of $n = 1, \ldots, N$ observations with p-dimensional feature vectors $\boldsymbol{x}_n^\top$ and binary labels $y_n \in 0, 1$, the probability of a positive event can be modeled with the *logistic function* and a single weight ω per feature. I.e., $p(Y_n = 1|\boldsymbol{x}_n, \boldsymbol{\omega}) = \sigma(\boldsymbol{x}_n^\top \boldsymbol{\omega}) = 1/(1 + \exp(-\boldsymbol{x}_n^\top \boldsymbol{\omega}))$, referred to as p_n in the following. The optimization problem for learning the weights $\boldsymbol{\omega}$ becomes

$$\min_{\boldsymbol{\omega}} \quad \Omega_{L_1}(\boldsymbol{\omega}) - \sum_{n=1}^{N} y_d \log(p_n) + (1 - y_d) \log(1 - p_n), \tag{1}$$

where $\Omega_{L_1} = \boldsymbol{\lambda}^\top |\boldsymbol{\omega}|_1 = \sum_{i=1}^{p} \lambda_i |\omega_i|$ is added to control overfitting and produce sparse solutions. For skewed target distributions, an intercept term ω_0 may be included in the model by appending an all-one feature to all observations. The corresponding regularization term λ_0 then needs to be fixed to zero.

For training the logistic regression model, one can use gradient-descent type optimizers and quasi-Newton based algorithms are a popular choice. With L_1-penalty, however, a little care must be taken since off-the-shelf Newton-based solvers require the objective function to be differentiable, which (1) is not due to the penalty function which is not differentiable in zero. In this work we base our logistic regression training on OWL-QN [20] for batch learning. For online learning using stochastic gradient descent with L_1-penalization, see [21].

Performing predictions with a logistic regression model is as simple as computing the logistic function on the features of a test observation, $\tilde{\boldsymbol{x}}$. In terms of speed, however, it matters how the features of $\tilde{\boldsymbol{x}}$ are represented. In particular for a binary feature vector $\boldsymbol{x}$

$$\sigma(\boldsymbol{x}^\top \boldsymbol{\omega}) = \frac{\exp(\boldsymbol{x}^\top \boldsymbol{\omega})}{1 + \exp(\boldsymbol{x}^\top \boldsymbol{\omega})} = \frac{\prod_{i=1}^{p} \exp(x_i \omega_i)}{1 + \prod_{i=1}^{p} \exp(x_i \omega_i)} \overset{\boldsymbol{x} \text{ binary}}{=} \frac{\prod_{i':x_{i'}=1} \exp(\omega_{i'})}{1 + \prod_{i':x_{i'}=1} \exp(\omega_{i'})} \tag{2}$$

I.e., predicting for binary feature vectors scales in the number of non-zero elements of the feature vector, which makes computations considerably faster. Additionally, using the right-hand side of (2), $\exp(\cdot)$ can be performed when storing the weights in memory or a database, hences saves further processing power. This has two consequences: 1) Binary features are more desirable for making real-time predictions and 2) the sparser the features, the less computation time and I/O from databases is required.

3 Experiments

The data we use for our experiments originate from Adform's ad transaction logs. In each transaction, e.g., when an ad is served, the URL where the ad is being displayed and a unique identifier of the users web browser is stored along with an identifier of the ad. Likewise, a transaction is logged when a user clicks an ad. From these logs, we prepare a data set over a period of time and use the final day for testing and use the rest for training.

As a pre-processing step, all URLs in the transaction log are stripped of any query-string that might be trailing the URL[1], however the log data are otherwise unprocessed.

3.1 Dimensionality reduction

From the training set transactions, we produce a binary bipartite graph of users in the first mode and URLs in the second mode. This is an unweighted, undirected graph where edges represent which URLs a user has seen, i.e., we do not use the number of times the user has engaged each URL. The graph we obtain has M=9,304,402 unique users and N=7,056,152 unique URLs. We denote this graph `UL`.

As we will be repeating numerous supervised learning experiments, that each can be quite time consuming for the entire training set, we do our main analysis based on experiments from a subset

[1] Query-string: Anything trailing an "?" in a URL, including the "?".

of transactions. As an inclusion criteria, we select the top M_{small} =99,854 users based on the number of URLs they have seen and URLs with visits from at least 100 unique users, resulting in N_{small} =70,436 URLs being included. Based on those subsets of users and URLs, we produce a smaller transaction log, from which we also construct a bipartite graph denoted $\texttt{UL}_{\text{small}}$.

3.1.1 Method details

For the sampled data for unsupervised learning, $\texttt{UL}_{\text{small}}$, we use the different dimensionality reduction techniques presented in Section 2 to obtain new per-user and per-URL features.

For obtaining the SVD-based dense left and right singular vectors, we use `SVDS` included with Matlab to compute the 500 largest eigenvalues with their corresponding eigenvectors. In the supervised learning, by joining our data by user and URL with the left and right singular vectors, respectively, we can use anything from 1 to 500 of the largest eigenvectors for each modality as features.

We use the NMF Matlab Toolbox from [22] to decompose $\texttt{UL}_{\text{small}}$ into non-negative factors. We use the original algorithm introduced in [3] with the least-squares objective and multiplicative updates (`nmfrule` option in the NMF Toolbox). With NMF we need to decide the model order, i.e., number of components to fit in each of the non-negative factors. Hence, to investigate the influence of NMF model order, we train NMF using various model orders of 100, 300, and 500 number of components. We run the toolbox with the default configurations for convergence tolerance and maximum number of iterations.

As detailed in Section 2.1.3, we use the GPU sampling scheme from [6] for massively speeding up the computation of the IRM model. The IRM estimation infers the number of components (i.e., clusters) separately for each modality, however, it does require we input a maximum number of components for users and URLs. For $\texttt{UL}_{\text{small}}$, we run with $K_{\max}$=500 for both modalities and terminate the estimation after 500 iterations. The IRM infers 216 user clusters and 175 URL cluster for $\texttt{UL}_{\text{small}}$, i.e., well below the $K_{\max}$ we specify.

For the full dataset `UL`, we have only completed the dimensionality reduction using IRM, which is thanks to our access to the aforementioned GPU sampling code. Again we run the IRM for 500 iterations, and with 500 as $K_{\max}$ for each modality. The IRM infers 408 user clusters and 380 URL clusters for `UL`; again well below $K_{\max}$.

Running the SVD and NMF for a data set the size of `UL` within acceptable times (i.e., within a day or less), is in it self a challenge and requires specialized software, either utilizing GPUs or distributed computation (or both). As we have not had immediate access to any implementations capable hereof, the SVD and NMF decompositions of `UL` remain as future work. Hence, for click-through rate prediction on the full data set, we demonstrate only the benefit of using the IRM cluster features over not using any dimensionality reduction.

3.2 Supervised learning

For testing the various dimensionality reductions, we construct several training and testing data sets from RTB logs with observations labeled as click (1) or non-click (0). The features we use are summarized in table 1.

Based on the full set of users and URLs as well as the sub-sampled sets, detailed in Section 3.1, we prepare training and testing data sets based on the features of Table 1 for our logistic regression classifier. We denote the full dataset `SL` and the sampled $\texttt{SL}_{\text{small}}$. The data are represented as $N \times p$ matrices, i.e., with columns being features and rows being observations.

3.2.1 Method details

From the predictors of Table 1, we train a number of logistic regression classifiers, using L_1-penalization for sparsity, see also Section 2.2. For the stopping criteria, we run until the change of the objective value between iterations falls below 1e-6. As the classes (clicks vs. non-clicks) are highly unbalanced, we also learn an unpenalized intercept term. In order not to introduce any advantages (or disadvantages) to some predictors over others, we do not normalize the input features for any of the predictors in any way. Rather, we first select one regularization strength, λ_{f_1}, for the baseline predictor only, f_1, and fix that through all other trials. In each experiment, we then

Table 1: Names and descriptions of the predictors used to predict click-through rates.

Ref	Feature(s)	Description
f_1	(BannerId, Url)	A one-of-K encoding of the cross-features between $BannerId$ and Url, which indicates where a request has been made. This serves as a baseline predictor in all of our experiments.
f_2	UrlsVisited	A vector representation (zeros and ones) of URLs that a specific user has visited in the past.
f_3	UserCluster	A one-of-K encoding of which IRM cluster a specific user belongs to.
f_4	UrlCluster	A one-of-K encoding of which IRM cluster a specific URL belongs to.
$f_5^{(n)}$	UserSVDnLoading	The continous-valued n-dimension left singular vector of a specific user from the SVD.
$f_6^{(n)}$	UrlSVDnLoading	The continous-valued n-dimension right singular vector of a specific URL from the SVD.
$f_7^{(n)}$	UserNMFnLoading	The continous-valued cluster assignment vector of a specific user according to the NMF-n decomposition.
$f_8^{(n)}$	UrlNMFnLoading	The continous-valued cluster assignment vector of a specific URL according to the NMF-n decomposition.

use other predictors f_3-f_8 in addition to f_1 and select another regularization strength, $\lambda_{f_{\geq 3}}$, jointly regularizing those predictors, but with λ_{f_1} still fixed for f_1. We compare to using f_2 regularized by λ_{f_2} in addition to f_1 and henceforth refer to this model as NODR, short for no dimensionality reduction.

For each trained model, we measure the performance in terms of the negative Bernoulli log-likelihood (LL), which measures the mismatch between the observations and the predictions of the model, i.e., the lower, the better. The likelihoods we report are normalized with respect to the baseline likelihood of the click-through rate evaluated on the test set, such that in order to outperform the baseline, they should fall between 0 and 1.

3.3 Results on $\texttt{SL}_{\text{small}}$

For the sampled data the number of observations are as follows: N_{train}=138,847 and N_{test}=4,273. In order to give the reader an idea about the dimensionalities of the features as well as their sparsity, in Table 2 we summarize some numbers on the predictors on the sampled data set. For features f_1,f_3, and f_4, the number of non-zeros (nnz) and sparsities are somewhat trivial, since these are categorical features represented as one-of-K binary vectors. For the SVD features, f_5 and f_6, we see that the feature vectors become completely dense. For the NMF features, however, we can confirm the methods' ability to produce sparse components, i.e., only between 20-33% of the components turn up as non-zeros, yet they are far from the sparsities of the IRM cluster features, f_3 and f_4.

In Table 3, we report the normalized likelihoods, lifts and test-set optimal regularization strengths λ_{f_1} and $\lambda_{f_{\geq 3}}$, with varying features used for training. The lifts are all relative to model f_1. The penalization strength $\lambda_{f_1} = 0.8$ is selected as the one maximizing the performance of the classifier using only f_1, and is kept fixed for all the other classifiers. Note, that generalization of the penalization terms is an issue we do not currently address. The time reported in the table are the

Table 2: Statistics of the various predictors on the sampled data set.

Feature	p	nnz	sparsity
f_1	44086	143120	1 - 2.3e-5
f_2	42910	8824491	1 - 1.4e-3
f_3	216	143120	1 - 4.6e-3
f_4	175	143120	1 - 5.7e-3
f_5,f_6	100 / 300 / 500	$dense$	0
f_7	100 / 300 / 500	4745568 / 9780078 / 13993847	0.67 / 0.77 / 0.80
f_8	100 / 300 / 500	4174552 / 14363612 / 23712222	0.71 / 0.67 / 0.67

Table 3: Results for the sub-sampled data set.

	Model	$(\lambda_{f_1}, \lambda_{f_2}, \lambda_{f_{\geq 3}})$	Time (s)	nnz_{all}	nnz_{f_2}	LL·100	% Lift
	f_1	(0.8, - , -)	9	3612	-	93.83	0.00
NODR	f_1, f_2	(0.8, 10.6, -)	91	3943	760	88.15	6.05
IRM	f_1, f_3	(0.8, - , 6.0e-4)	13	3653	-	90.19	3.88
	f_1, f_3, f_4	(0.8, - , 7.0e-4)	16	3674	-	**89.84**	**4.25** ∇
	$\nabla + f_2$	(0.8, 15.4, 7.0e-4)	76	3861	366	87.78	6.45
SVD	$f_1, f_5^{(100)}$	(0.8, - , 0.1)	19	3479	-	89.87	4.22
	$f_1, f_5^{(300)}$	(0.8, - , 0.3)	29	3502	-	89.73	4.37
	$f_1, f_5^{(500)}$	(0.8, - , 0.3)	56	3552	-	89.73	4.37
	$f_1, f_5^{(100)}, f_6^{(100)}$	(0.8, - , 7.0e-4)	649	3409	-	89.15	4.99
	$f_1, f_5^{(300)}, f_6^{(300)}$	(0.8, - , 7.0e-4)	2487	3702	-	**88.92**	**5.23** $\diamond$
	$f_1, f_5^{(500)}, f_6^{(500)}$	(0.8, - , 1.2e-3)	4082	4027	-	89.55	4.56
	$\diamond + f_2$	(0.8, 10.8, 7.0e-4)	3291	4063	484	87.90	6.32
NMF	$f_1, f_7^{(100)}$	(0.8, - , 6.0e-3)	30	3453	-	89.38	4.74
	$f_1, f_7^{(300)}$	(0.8, - , 3.0e-3)	40	3467	-	89.15	4.99
	$f_1, f_7^{(500)}$	(0.8, - , 2.0e-3)	45	3521	-	88.68	5.49
	$f_1, f_7^{(100)}, f_8^{(100)}$	(0.8, - , 5.0e-3)	151	3389	-	89.05	5.09
	$f_1, f_7^{(300)}, f_8^{(300)}$	(0.8, - , 6.0e-3)	392	3468	-	**87.89**	**6.33** $\circ$
	$f_1, f_7^{(500)}, f_8^{(500)}$	(0.8, - , 4.0e-3)	740	3635	-	93.59	0.26
	$\circ + f_2$	(0.8, 11.2, 6.0e-3)	641	3973	680	86.91	7.38

seconds it takes to train the logistic regression classifier. nnz_{all} and nnz_{f_2} are the respective number of non-zero weights of the resulting classifier for all the features and the f_2 feature only

In order to be able to further elaborate on the pros and cons of using the various dimensionality reduction techniques as features in the logistic regression classifier, we carry out another set of experiments for the models highlighted (bold and marked $\nabla, \diamond, \circ$) in Table 3. We fix the values of λ_{f_1} and $\lambda_{f_{\geq 3}}$ to the values from $\nabla, \diamond$, and $\circ$, respectively, and append f_2 as an additional feature with each model and then tune the regularization strength λ_{f_2}. The results are shown in the rows of Table 3 with the symbols $\nabla + f_2, \diamond + f_2$, and $\circ + f_2$ under "Model".

The final experiment we run is with the full data set where we only evaluate the IRM based features and compare those to not using any dimensionality reduction. The number of observations for train and test are N_{train}=5,460,229 and N_{test}=188,867. The selection of regularization terms we do as in the previous experiments. The results are reported in Table 4.

Table 4: Results for the full data set.

	Model	$(\lambda_{f_1}, \lambda_{f_2}, \lambda_{f_{\geq 3}})$	Time (s)	nnz_{all}	nnz_{f_2}	LL·100	% Lift
	f_1	(0.7, - , -)	34	14152	-	91.76	0.00
NODR	f_1, f_2	(0.7, 10.2, -)	195	15673	3010	88.71	3.32
IRM	f_1, f_3, f_4	(0.7, - , 1.2e-3)	51	13604	-	89.35	2.63 ∇
	$\nabla + f_2$	(0.7, 10.2, 1.2e-3)	293	16018	2939	88.19	3.89

4 Discussion

From Table 3 we first concentrate on the best models from each dimensionality reduction, i.e., the results highlighted in bold. Comparing the lifts, we see that the NMF-300 features perform roughly one %-point better than the SVD-300 features, which then in turn perform roughly another %-point better than the IRM cluster features. Comparing to the classifier using just f_1 and f_2, i.e., no dimensionality reduction, we see that only the NMF-based classifier achieves slightly higher lift. Hence, using SVD or IRM based features as *a replacement* for the f_2 feature would result in worse predictions. Seeing the number of non-zero weights dropping from 3943 using f_2 to 3468 using both NMF-300 features, indicates that the NMF offers a more economical representation which can replace f_2 while not sacrificing performance. The performance gain of NMF-300 we expect is achieved by the implicit data grouping effects of NMF, i.e., recommender effects.

In terms of training speed, we see that while the IRM based features fare worst in terms of lift, the fact that each mode is a categorical value represented in a one-of-K binary vector makes the input matrix very sparse, which speeds up the training of our classifier significantly and the model trains at least an order of magnitude faster than the other dimensionality reduction techniques and even significantly faster than training the NODR model. Hence, if fast training is a priority, either no dimensionality reduction should be used or the IRM based features can be used, but at the cost of slightly lower lift.

We now turn to the results for the models $\nabla + f_2, \diamond + f_2$, and $\circ + f_2$ in Table 3. Here we investigate how the learning of weights for the high-cardinality feature f_2 is affected when combined with each of the optimal settings from the reduced dimension experiments. Again, observing the lifts, the NMF-300 based features combined with f_2 obtains the highest lift. However, the IRM based features now outperform the SVD ones and using either of the techniques in combination with f_2, we are able to obtain higher lifts than using only f_2.

For the training speed, we again see that the training using IRM features is by far the fastest amongst SVD and NMF and it is still faster than using f_2 only. What is more interesting, is the resulting number of non-zero weights, both in total and in the f_2 feature alone. Of all the different dimensionality reductions as well as NODR, using the IRM based representation requires the fewest non-zero weights at its optimal settings. Additionally, recalling from Section 2.2, that predictions can be made computationally very efficient, when the input features are binary indicator vectors, the IRM becomes all the more tractable. By combining the IRM based features with the explicit predictors f_1 and f_2, our classifier is able to improve the lift over not using dimensionality reduction while reducing the need for fetching many weights for predictions and with only a small reduction in lift, compared to the more computationally expensive classifiers based on NMF and SVD.

Finally, in Table 4 we have run experiments using just the IRM based predictors with the full data set. The results confirm our findings from Table 3 and at the same time demonstrates both the feasibility of processing very large bipartite graphs using IRM as well as the application of the user and URL clusters as predictors of click-through rates.

5 Conclusion

We have presented results that demonstrate the use of three bimodal dimensionality reduction techniques, SVD, NMF, and IRM, and their applications as predictors in a click-through rate data set. We show that the compact representation based on the NMF is, in terms of predictive performance, the best option. For applications where fast predictions are required, however, we show that the binary representation from the IRM model is a viable alternative. The IRM based predictors yield the fastest training speed in the supervised learning stage, produces the most sparse model and offers the fastest computations at run-time, while incurring only a limited loss of lift relative to the NMF. In applications such as real-time bidding, where fast database I/O and few computations are key to success, we recommend using IRM based features as predictors.

References

[1] C Apte, E Bibelnieks, and R Natarajan. Segmentation-based modeling for advanced targeted marketing. *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, 2001.

[2] G Golub and W Kahan. Calculating the singular values and pseudo-inverse of a matrix. *Journal of the Society for Industrial & Applied Mathematics*, 1965.

[3] D.D. Lee and H.S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999.

[4] C. Kemp, J.B. Tenenbaum, T.L. Griffiths, T. Yamada, and N. Ueda. Learning systems of concepts with an infinite relational model. In *Artificial Intelligence, Proceedings of the National AAAI Conference on*, 2006.

[5] D Hillard, E Manavoglu, H Raghavan, C Leggetter, E Cantú-Paz, and R Iyer. The sum of its parts: reducing sparsity in click estimation with query segments. *Information Retrieval*, 14(3):315–336, February 2011.

[6] TJ Hansen, M Morup, and LK Hansen. Non-parametric co-clustering of large scale sparse bipartite networks on the GPU. *Machine Learning for Signal Processing (MLSP), 2011 IEEE International Workshop on*, 2011.

[7] KS Dave and V Varma. Learning the click-through rate for rare/new ads from similar ads. *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 897–898, 2010.

[8] M Richardson, E Dominowska, and R Ragno. Predicting clicks: estimating the click-through rate for new ads. *Proceedings of the 16th international conference on World Wide Web*, pages 521–530, 2007.

[9] AK Menon, KP Chitrapura, S Garg, D Agarwal, and N Kota. Response prediction using collaborative filtering with hierarchies and side-information. *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '11*, page 141, 2011.

[10] Y Chen and M Kapralov. Factor modeling for advertisement targeting. *Advances in Neural Information Processing Systems*, 2009.

[11] MW Berry and M Browne. Email surveillance using non-negative matrix factorization. *Computational & Mathematical Organization Theory*, 1(11.3):249–264, 2005.

[12] W Xu, X Liu, and Y Gong. Document clustering based on non-negative matrix factorization. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, volume pages, page 273. ACM, 2003.

[13] BO Wahlgreen and LK Hansen. Large scale topic modeling made practical. In *Machine Learning for Signal Processing (MLSP), 2011 IEEE International Workshop on*, volume 1, pages 1–6. IEEE, September 2011.

[14] QW Dong, XL Wang, and L Lin. Application of latent semantic analysis to protein remote homology detection. *Bioinformatics*, 22(3):285–290, 2006.

[15] Zhao Xu, Volker Tresp, Kai Yu, and Hans-Peter Kriegel. Learning infinite hidden relational models. *Uncertainity in Artificial Intelligence (UAI2006)*, 2006.

[16] K. Nowicki and T.A.B. Snijders. Estimation and prediction for stochastic blockstructures. *Journal of the American Statistical Association*, 96(455):1077–1087, 2001.

[17] R.M. Neal. Markov chain sampling methods for dirichlet process mixture models. *Computational and Graphical Statistics, Journal of*, 9:249–265, 2000.

[18] S. Yu K. Yu H.-P. Kriegel Z. Xu, V.Tresp. Fast inference in inifinite hidden relational models. *Mining and Learning with Graphs (MLG'07)*, 2007.

[19] CM Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 2007.

[20] G Andrew and J Gao. Scalable training of L 1-regularized log-linear models. *Proceedings of the 24th international conference on Machine learning*, pages 33–40, 2007.

[21] Y Tsuruoka, J Tsujii, and S Ananiadou. Stochastic gradient descent training for L1-regularized log-linear models with cumulative penalty. *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, 1:477, 2009.

[22] Y Li and A Ngom. The non-negative matrix factorization toolbox for biological data mining. *Source code for biology and medicine*, 8(1):10, January 2013.

APPENDIX E

Paper 3

2014 IEEE INTERNATIONAL WORKSHOP ON MACHINE LEARNING FOR SIGNAL PROCESSING, SEPT. 21–24, 2014, REIMS, FRANCE

COMPACT WEB BROWSING PROFILES FOR CLICK-THROUGH RATE PREDICTION

Bjarne Ørum Fruergaard

Adform ApS
Hovedvagtsgade 6
DK-1103 Copenhagen K, Denmark
b.fruergaard@adform.com

Lars Kai Hansen

Technical University of Denmark
Anker Engelunds Vej 1,
DK-2800 Kgs. Lyngby, Denmark
lkai@dtu.dk

ABSTRACT

In real time advertising we are interested in finding features that improve click-through rate prediction. One source of available information is the bipartite graph of websites previously engaged by identifiable users. In this work, we investigate three different decompositions of such a graph with varying degrees of sparsity in the representations. The decompositions that we consider are SVD, NMF, and IRM. To quantify the utility, we measure the performances of these representations when used as features in a sparse logistic regression model for click-through rate prediction. We recommend the IRM bipartite clustering features as they provide the most compact representation of browsing patterns and yield the best performance.

1. INTRODUCTION

In online display advertising the traditional way of serving ads based on fixed price deals between publishers and advertisers has recently given way to the increasingly popular real time bidding (RTB) ad exchanges. The concept of RTB is that, upon a user's request for a specific URL, a real-time auction is started amongst advertisers, competing to serve their advertisement. The participants are allotted a limited time on the order of 100ms to query their data sources and come up with a bid, and the winner's advertisement is displayed. From the advertisers perspective, estimating the value of showing an ad is therefore of crucial importance. In this work we focus on click-through rate (CTR) prediction, i.e., the value of interest is the estimated probability of click, however, our approach generalizes to any measure of value, such as lead or conversion rate.

A challenge when dealing with click-through data, is that many URLs and users have never generated clicks for the ad at hand, and there are no direct empirical statistics of the CTR to base a prediction on. Instead, we may resort to side-information (*features*) about the URL and user and try to find common traits or behaviors which generalize to predictions of clicks. We investigate the usefulness of behavioral features derived from analyzing previous user-URL engagements, represented as a bipartite binary graph, where a link exists between a user and a URL, if the user has visited the URL. Based on this graph, we build URL and user *profiles*, which we then use as features in a CTR prediction setup. By using features derived from web-browsing patterns, the assumption is that similar profiles also have similar click statistics.

With the number of active URLs and Internet users in the ranges of millions and billions, respectively, the representation of the profiles has a large impact on the requirements for computation, storage and retrieval. One model which can compactly represent profiles of URLs and users in the aforementioned graph, is the Infinite Relational Model (IRM), which simultaneously clusters the users and URLs into single-membership segments. We investigate how this model competes against two other techniques for computing profiles based on the Non-negative Matrix Factorization (NMF) and the Singular Value Decomposition (SVD); the former is known to produce sparse components [1], whereas the latter produces dense vector profiles.

As both the IRM and the NMF are randomized methods with non-convex objective functions, we experiment with multiple restarts and different parameterizations. In particular for the IRM, which has several hyper-parameters we need to manually set, we find that some experimentation is required in order to get good downstream CTR performance.

Comparing the techniques at their optimal settings using CTR performances as measures, we find that the IRM model performs superior to both the NMF and the SVD. We also observe that URL profiles alone yield more performance than user profiles alone. In summary, we recommend to use IRM based profiles of web browsing behavior, which offers both performance as well as the practical advantages of being compact features.

This paper is organized as follows: In Section 2 we introduce the bipartite graph decomposition techniques (2.1) and the model for predicting click-through rates (2.2). In Section 3 we describe our experimental setup for evaluating each of the graph decompositions in terms of their respective perfor-

978-1-4799-3694-6/14/$31.00

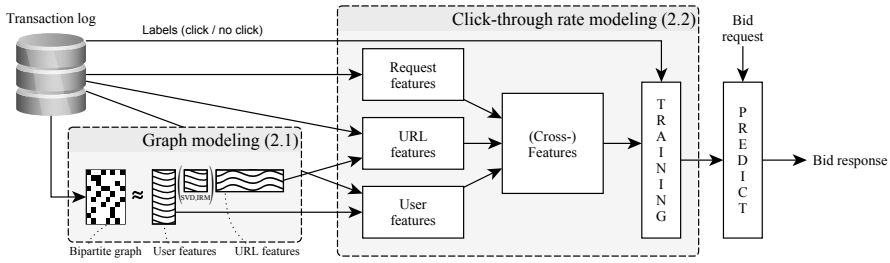


Fig. 1. Proposed design of a pipeline for click-through rate prediction. The "wavy" matrices in the graph modeling part (Section 2.1) change depending on decomposition technique. The features passed on to the click-through rate modeling task from each of the different decompositions are vectors from the User and Url components, respectively. I.e., the User feature passed on is the transpose of the row-vector corresponding to the user of a training data observation and the Url feature is the column-vector corresponding to the Url of the observation.

mances and present the results. The results are discussed in Section 4 and finally, we conclude on our work in Section 5.

1.1. Related work

This work is a continuation of [2], where we first proposed the use of the IRM as a dimensionality reduction technique for the user-by-URL graph. In this paper, we carry out extensive experiments with the hyper-parameters of the IRM model as well as run several restarts with both the IRM and NMF. Apart from this, we also take a different approach to regularization in our CTR model. Hence, all the results presented herein are new. More importantly, the new improved evaluation leads to a revised conclusion, namely to recommend IRM over NMF both in terms of prediction performance and as earlier in terms of compactness/sparsity of the representation.

For general references to collaborative filtering, using approximate user profiles, as well as training a probabilistic model based on logistic regression for computational advertising, we refer to our first paper [2].

To our knowledge, [2] is the first work to demonstrate the IRM model as a dimensionality reduction technique for producing features for a probabilistic model. The approach bears some similarity with the Infinite Hidden Relational Model (IHRM) [3], the main difference being that with IHRM applied to click-through data, the latent cluster variables model coherences in click-patterns informed by node attributes (e.g., websites visited), whereas in our approach the latent variables are unrelated to clicks and instead model coherent structure in browsing patterns (i.e., websites visited, not clicks). Our two-step approach has the advantages that 1) inference is simpler and 2) the clusters are not tailored for CTR prediction and can be incorporated for other analytic purposes. In terms of downstream performance, we do not have IHRM results to compare against, the main reason being that there is no implementation available of IHRM suitable for CTR prediction.

2. METHODS

In Figure 1 we show our proposed setup, i.e., a system for click-through rate prediction in two steps: Graph modeling (Section 2.1) and Click-through rate modeling (Section 2.2).

2.1. Graph modeling

In the following, we repeat the introduction of each of the decomposition techniques from [2] with only minor modifications.

Let a bipartite graph for I users and J URLs be represented as an adjacency matrix $\boldsymbol{A}$ with dimensions $I \times J$. A link, $A_{ij} = 1$, means user i has visited URL j.

2.1.1. Singular value decomposition

The singular value decomposition (SVD) of a rank R matrix $\boldsymbol{A}$ is given as the factorization $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top = \sum_{i=1}^{R} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^\top$, where $\boldsymbol{U}$ and $\boldsymbol{V}$ are unitary matrices $\boldsymbol{U}^\top\boldsymbol{U} = \boldsymbol{V}^\top\boldsymbol{V} = \boldsymbol{I}$ and hold the left and right singular vectors of $\boldsymbol{A}$, respectively. The diagonal matrix $\boldsymbol{\Sigma}$ contains the singular values, σ_i, of $\boldsymbol{A}$. By selecting only the K largest singular values of $\boldsymbol{\Sigma}$, i.e., truncating all other singular values to zero, one obtains the approximation $\tilde{\boldsymbol{A}} = \boldsymbol{U}\tilde{\boldsymbol{\Sigma}}\boldsymbol{V}^\top = \sum_{i=1}^{K} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^\top$, which is the rank K optimal solution to $\arg\min ||\boldsymbol{A} - \tilde{\boldsymbol{A}}||_2^2$. This truncation corresponds to disregarding the $R-K$ dimensions with the least variances as noise.

2.1.2. Non-negative matrix factorization

NMF [1] is a matrix factorization similar to SVD, the crucial difference being that NMF decomposes into non-negative factors and imposes no orthogonality constraints. Given a non-negative input matrix $\boldsymbol{A}$ with dimensions $I \times J$, NMF approximates $\boldsymbol{A} \approx \boldsymbol{W}\boldsymbol{H}$, where $\boldsymbol{W}$ is an $I \times K$ non-negative

matrix, $\boldsymbol{H}$ a $K \times J$ non-negative matrix, and K is the number of components. By selecting $K << min(I, J)$ one approximates $\boldsymbol{A}^{(I\times J)} = \boldsymbol{W}^{(I\times K)}\boldsymbol{H}^{(K\times J)} + \boldsymbol{E}^{(I\times J)}$, thereby discarding the residual (unconstrained) matrix $\boldsymbol{E}$ as noise.

NMF has achieved good empirical results as an unsupervised learning technique within many applications, e.g., for document clustering [4, 5, 6], visual coding [1], and bioinformatics [7]. In [8] NMF is used for computational advertising.

2.1.3. Infinite relational model

The Infinite Relational Model (IRM) is a Bayesian generative model for graphs and it can be applied as a co-clustering approach for bipartite networks, where the nodes of each mode are grouped simultaneously. Compared to existing co-clustering approaches, the IRM is preferred as it implements the properties of binary graphs and allows the number of components of each mode to be inferred from the data.

The generative process for the Relational Model [9, 3, 10] is given by:

$$\begin{aligned}
\boldsymbol{\mu}^{(1)} &\sim Dirichlet(\alpha^{(1)}/K^{(1)}\boldsymbol{e}^{(1)}) && \texttt{row cluster prob.}\\
z_i^{(1)} &\sim Multinomial(\boldsymbol{\mu}^{(1)}) && \texttt{row cluster assignment}\\
\boldsymbol{\mu}^{(2)} &\sim Dirichlet(\alpha^{(2)}/K^{(2)}\boldsymbol{e}^{(2)}) && \texttt{col. cluster prob.}\\
z_j^{(2)} &\sim Multinomial(\boldsymbol{\mu}^{(2)}) && \texttt{col. cluster assignment}\\
\eta_{mn} &\sim Beta(\beta^{+}, \beta^{-}) && \texttt{between cluster mixing}\\
A_{ij} &\sim Bernoulli(\boldsymbol{z}_i^{(1)\top}\boldsymbol{\eta}\boldsymbol{z}_j^{(2)}) && \texttt{generate links}
\end{aligned}$$

For $i = 1, \ldots, I$, $j = 1, \ldots, J$, $m = 1, \ldots, K^{(1)}$, and $n = 1, \ldots, K^{(2)}$. $K^{(1)}$ and $K^{(2)}$ denote the number of row and column clusters respectively whereas $\boldsymbol{e}^{(1)}$ and $\boldsymbol{e}^{(2)}$ are vectors of ones of size $K^{(1)}$ and $K^{(2)}$. $\boldsymbol{z}_i^{(1)}$ and $\boldsymbol{z}_j^{(2)}$ denote 1-of-K encoded binary vectors of $z_i^{(1)}$ and $z_j^{(2)}$, respectively, and is a convenient representation for formulation as well as implementation. The limits $K^{(1)} \to \infty$ and $K^{(2)} \to \infty$ lead to the Infinite Relational Model (IRM) which has an analytic distribution given by the Chinese Restaurant Process (CRP) [3, 9, 11].

Rather than collapsing the parameters of the model, we apply blocked sampling that allows for parallel GPU computation [12]. Moreover, the CRP is approximated by the truncated stick breaking construction (TSB), and the truncation error becomes insignificant when the model is estimated for large values of $K^{(1)}$ and $K^{(2)}$, see also [13].

2.2. Click-through rate prediction model

For learning a predictive model of click-through rates based on historical data, we employ logistic regression with sparsity constraints; for further details see for instance [14, 15]. Given data consisting of $n = 1, \ldots, N$ observations with p-dimensional feature vectors $\boldsymbol{x}_n$ and labels $y_n \in \{-1, 1\}$, the probability of a positive event can be modeled with the *logistic function* and a single weight ω per feature. I.e., $p(Y_n = 1|\boldsymbol{x}_n, \boldsymbol{\omega}) = \sigma(\langle \boldsymbol{x}_n, \boldsymbol{\omega}\rangle) = 1/(1 + \exp(-\langle \boldsymbol{x}_n, \boldsymbol{\omega}\rangle))$, where $\langle\cdot\rangle$ denotes the dot-product. The optimization problem for learning the weights $\boldsymbol{\omega}$ becomes

$$\min_{\boldsymbol{\omega}} \ \Omega_{L_1}(\boldsymbol{\omega}) + \sum_{n=1}^{N} \log(1 + \exp(-\langle y_n \boldsymbol{x}_n, \boldsymbol{\omega}\rangle)), \quad (1)$$

where $\Omega_{L_1} = \boldsymbol{\lambda}^\top|\boldsymbol{\omega}|_1 = \sum_{i=1}^{p} \lambda_i|\omega_i|$ is added to control overfitting and produce sparse solutions. To accommodate skewed target distributions, we add an intercept term ω_0 in the model by appending an all-one feature to all observations. The corresponding regularization term λ_0 is fixed to zero.

Due to the non-differentiability at the origin of the penalty function, we employ a batch learning optimizer based on OWL-QN [15]. With very large datasets, online learning can be achieved using stochastic gradient descent. For details, see [16].

As we note in [2], computation of predictions using the model are considerably faster, when the features are binary; hence, a beneficial side-effect of using the IRM for building profiles is, that it allows for faster computation of predictions. I.e., computing the logistic function for a dense profile vector of size m scales as $\mathcal{O}(m)$, whereas a binary one-of-K encoded clustering profile scales as $\mathcal{O}(1)$ and only involves addition.

Since the input data we will use for training the logistic regression model has a heterogeneous structure, e.g., some features are categorical (encoded as binary one-of-K), others very high-cardinality indicator vectors (binary bag-of-features), and some continuous valued vectors, in order for high-energy features not to suppress other, low-energy features, we need to normalize input data. With an intercept term, we do not need to mean-subtract, but we will apply (the equivalent of) a scaling of each vector representing a feature to euclidean unit length on average. A simple trick, however, to avoid scaling the data, is to apply the inverse scaling coefficients element-wise to the global penalization weights $\boldsymbol{\lambda}$ (all identical, except λ_0) instead, thereby obtaining different per-feature penalization strengths.

3. EXPERIMENTS AND RESULTS

The data sets we use in our experiments originate from Adform's ad transaction logs. In each transaction, e.g., when an ad is served, the URL where the ad is being displayed and a unique identifier of the users web browser is stored along with an identifier of the ad. Likewise, a transaction is logged when a user clicks an ad. From these logs, we prepare a data set over a period of time and use the final day data for testing and the rest for training. Since we run many repeated experiments with random restarts and different parameterizations, we do not consider all URLs and all users in the transaction log, but limit ourselves to only the most frequent (M=99,854) users and (N=70,436) URLs.

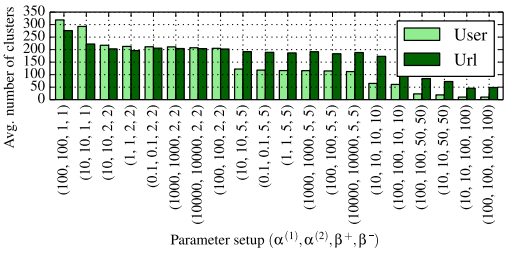


Fig. 2. Average number of clusters after 5 random restarts for the IRM model with the parameters set up according to the x-axis. All the averages are maximum ± 7, except (100,100,1,1) which is maximum ± 30.

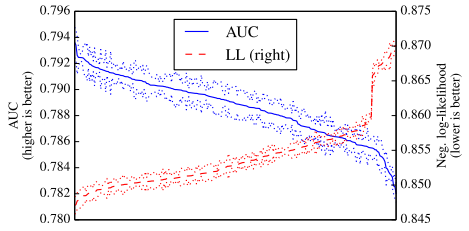


Fig. 3. Click prediction performance in terms of AUC and log-loss, sorted from best to worst. The hyper-parameters of the IRM as well as the penalization strength in the logistic regression are varied as described in Section 3. Since there are too many configurations to show in a meaningful way, the x-axis is hidden. The dotted lines represent standard deviations of the mean based on five random restarts.

As a pre-processing step, all URLs in the transaction log are stripped of any query-string (everything including and after an "?") that might be trailing the URL.

3.1. Graph modeling

From the training set transactions, we produce a $I \times J$ binary bipartite graph with users in the first mode and URLs in the second mode. This is an unweighted, undirected graph where links represent which URLs a user has seen, without any information about clicks. The total number of links in the graph is 6,648,174.

3.1.1. Method details

We use the dimensionality reduction techniques presented in Section 2.1 to obtain new per-user and per-URL features.

The SVD-based dense left and right singular vectors, are obtained with `SVDS` included with Matlab to compute the 500 largest eigenvalues with their corresponding eigenvectors. In the CTR modeling, by joining our data by user and URL with the left and right singular vectors, respectively, we can use anything from 1 to 500 of the largest eigenvectors for each modality as features. We experiment with 100, 300, and 500 components for the SVD.

We use the NMF decomposition in GraphLab [17]. We run NMF with 100, 300, and 500 components, with five random restarts and 500 iterations, saving each of the obtained models. Running on an Amazon EC2 instance with 32 Intel Xeon E5-2670 v2 cores @ 2.5GHz (r3.8xlarge), GraphLab utilizes 30 cores for the estimation of NMF and with 500 components and 500 iterations, the wall clock time for estimation is 11-12 minutes.

For speeding up the inference of the IRM, we use the GPU accelerated sampling scheme of [12]. The inference is run on an Intel Core i7-3820 CPU @ 3.60GHz with an Nvidia Tesla K20c (GK110 @ 706MHz, 13 SMPs, 2496 shader units, 5GB GDDR5 @ 2x1300MHz) running CUDA 5.5. Our code utilizes just one CPU and one GPU, although further speed-up can be achieved by parallelizing the host (CPU) code as well as distributing inference of $z_i^{(1)}$ and $z_j^{(2)}$ across several GPUs. Running 500 iterations allowing up to 500 clusters in each modality, takes 8-9 minutes. Without the GPU accelerated inference, the running time for 500 iterations is 49-50 minutes.

In order to understand the roles of the IRM hyper-parameters, we vary $\alpha^{(1)} = \alpha^{(2)} = \{0.1, 1, 10, 100, 1000, 10000\}$ and $\beta^{+} = \beta^{-} = \{1, 2, 5, 10, 50, 100\}$, with five random restarts for each parameterization. In order not to run inference for all combinations of hyper-parameters exhaustively, we initially run with $\alpha^{(1)} = \alpha^{(2)} = 10$, trying all values of β^{+} and β^{-}, as well as $\beta^{+} = \beta^{-} = 2$, trying all values of $\alpha^{(1)}$ and $\alpha^{(2)}$. After measuring click-through rate performances, using the optimal β's given fixed α's and the optimal α's given fixed β's, we then run another full sweep of the α's for fixed β's and β's for fixed α's, respectively. Hence, we end up inferring the IRM for each value of β's using α's=$\{10, 100\}$ and for each value of α's using β's=$\{2, 5\}$.

Although IRM infers the number of components from data, practicality requires we input a maximum number of components. We use $K_{\max}$=500 for both modalities and terminate the estimation after 500 iterations.

For the NMF and IRM decomposition, we run 20 additional random restarts with what we find as the optimal parameterizations.

3.2. Click-through rate prediction model

For testing the various dimensionality reductions, we construct several training and testing data sets from RTB logs with observations labeled as click or non-click. The features we use are: Up to 50,000 request URLs (one-of-K), ID of the banner (one-of-K), Top 50,000 domains visited in the past (binary bag-of-features), and the features from each of the investigated decompositions. We train models

$\alpha^{(1)}$	$\alpha^{(2)}$	β^+	β^-	λ	LL	AUC
10.0	10.0	5	5	1.5	0.8532 ± 0.0011	0.7936 ± 0.0012
10.0	10.0	5	5	2.0	0.8500 ± 0.0010	0.7934 ± 0.0012
10.0	**10.0**	**5**	**5**	**3.5**	$\mathbf{0.8472 \pm 0.0013}$	$\mathbf{0.7926 \pm 0.0014}$
0.1	0.1	2	2	2.0	0.8515 ± 0.0013	0.7925 ± 0.0008
1.0	1.0	5	5	1.5	0.8547 ± 0.0002	0.7925 ± 0.0004
100.0	100.0	2	2	2.0	0.8507 ± 0.0011	0.7924 ± 0.0003

Table 1. The top 6 performing configurations of the IRM experiments using both profiles in terms of negative Bernoulli log-likelihood. For the LL and AUC columns, the mean values with standard deviation of the mean are listed based on five random restarts.

for each decomposition and each trial adding the URL profile ("Url only"), the user profile ("User only"), and both ("Both"). We vary the global L_1 regularization strength $\lambda = \{1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0, 6.0, 7.0, 8.0, 9.0\}$.

For every model, we measure the performance on the final day data (held-out) in terms of the log-loss (LL) as well as AUC. LL measures the mismatch between the observations and the predictions of the model, i.e., the lower, the better. The likelihoods we report are normalized, such that in order to outperform the baseline, they should fall between 0 and 1. The LL, however, has the disadvantages of being sensitive to class imbalance as well as mis-calibration, which the AUC is invariant to. The AUC measures the expected proportion of positives ranked before a uniformly drawn random negative, thus should fall between 0.5 and 1.0 for a model to perform better than random and the closer to 1.0, the better. As our preference for a performance measure for model comparison, we lean towards the AUC.

3.3. Results

In order to investigate the effects of changing the hyper-parameters of the IRM, we show in Figure 2 the average number of clusters in each modality after 5 random restarts in each of in total 20 variations of parameters.

We document the impact on click-through rate performance in Figure 3, as we vary the IRM parameters as well as regularization strength λ. We supplement this with Table 1, showing the top 6 experiments ranked according to decreasing AUC. The settings we use for running additional experiments using the IRM are the ones in the row highlighted in bold, chosen because they show good performance in both measures.

For the NMF with each variation of features ("Url only", "User only", and "Both") we see a slight trend towards higher model orders in both performance measures and with $\lambda = 3.5$ pretty consistently ranking in the top. Therefore for subsequent analysis of the NMF features we use $k = 500$ and $\lambda = 3.5$.

The results vary more for the SVD depending on the model order, features and the performance measure. Furthermore, as the SVD is a unique model, we do not repeat experiments. Instead, all the results we report for the SVD features use the settings that optimize each measure for each variation of the features. For optimal AUC scores: "User only", $(k = 500, \lambda = 1.5)$; "Url only", $(k = 100, \lambda = 3.5)$; "Both", $(k = 100, \lambda = 1.5)$. Optimal LL: "User only", $(k = 300, \lambda = 3.5)$; "Url only", $(k = 100, \lambda = 3.5)$; "Both", $(k = 500, \lambda = 3.5)$.

With the best settings for the NMF and IRM, we run additional 20 random restarts (25 total) in order to get more certainty about the performances. In Figure 4 we summarize the results of those experiments.

4. DISCUSSION

The effects on the inferred number of components, with varying α's and β's is seen in Figure 2. We see that with the parameter sweeps that we have selected, the model orders vary from approximately 300 to less than 50 in each modality, all well below $K_{\max} = 500$, hence we do not expect this choice to have had a major impact on the inference.

Looking at Figure 3, we see that in terms of both performance measures, the hyper-parameter settings of the IRM (and λ) affect the downstream click-through rate predictions. From Table 1 as well as Figure 2, we see that the β parameters are the most critical to optimize. An interesting prospect for further analysis would be an extension of our IRM inference procedure to infer hyper-parameters automatically from data. Further discussion hereof is however outside the scope of this contribution.

In Figure 4(a-b) we emphasize two results: 1) Except for the SVD, there are bigger gains using URL profiles over user profiles (see white and light green/gray bars) and 2) using both profiles is better than using either alone (dark green/gray bars always best).

The box plots Figure 4(c-d) compare the IRM and NMF results. With respect to AUC for each combination of features, the IRM outperforms the NMF counterparts, i.e., notches are non-overlapping. For LL, however, the notches do overlap, so we cannot say with the same certainty, that IRM outperforms NMF. We report the same observation using a two-tailed t-test between (IRM, Both) and (NMF, Both): For AUC we get $p = 0.02$, but for LL we get $p = 0.13$.

It is interesting to note the differences between AUC and LL; as mentioned LL suffers from several drawbacks. Since class-imbalance as well as mis-calibrated probabilities due to regularization affects the log-loss, but not the AUC, we expect that a post-calibration technique could help alleviate the LL results.

5. CONCLUSION

The use of the Infinite Relational Model as a representation of URL and User profiles from Web browsing patterns shows

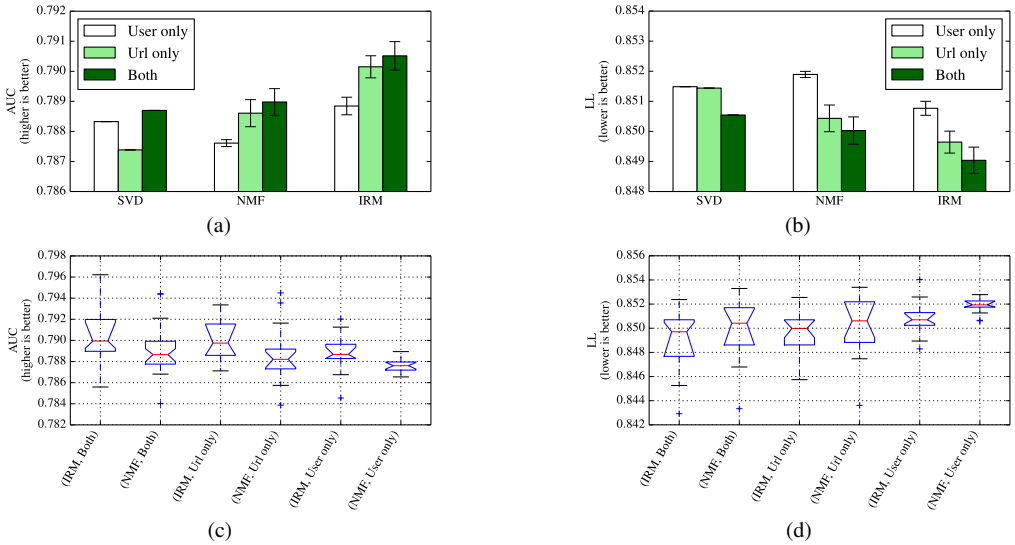


Fig. 4. (a-b) Average click performances in AUC (a) and LL (b) for the different decompositions (at optimal settings) and using only user (white), only URL (light green/gray), or both profiles (dark green/gray) as predictors. The error bars show $\pm$ one standard deviation from the mean based on 25 random restarts. (c-d) Box plots of the 25 random restarts for IRM and NMF performances, AUC (c) and LL (d). Box notches are 95% confidence interval estimates from 10k sample bootstrapping.

promising downstream performance in a click-through rate prediction model. Furthermore, the compact representation is advantageous for I/O and computational reasons, which make it applicable in a high-speed, high-throughput big data setting, such as real time bidding. We emphasize our results with experiments using more dense counterparts for building profiles, based on Singular Value Decomposition and Non-negative Matrix Factorization, that show that profiles based on the IRM are superior.

The results are particularly convincing when comparing models based on AUC. However, with performance measured in log-loss, we are underpowered, hence a future direction would be to test whether post-calibration might ameliorate the log-loss results.

Another direction we find very interesting, is to model a multigraph instead, i.e., with (multiple) links corresponding to (User, URL) frequencies, which can be done by exchanging the sampling distribution of the IRM with the Poisson instead of the Bernoulli.

6. REFERENCES

[1] D.D. Lee and H.S. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, no. 6755, pp. 788–791, 1999.

[2] BØ Fruergaard, TJ Hansen, and LK Hansen, "Dimensionality reduction for click-through rate prediction: Dense versus sparse representation," *arXiv preprint arXiv:1311.6976*, 2013, Unpublished manuscript.

[3] Z Xu, V Tresp, K Yu, and HP Kriegel, "Learning infinite hidden relational models," *Uncertainity in Artificial Intelligence*, 2006.

[4] MW Berry and M Browne, "Email surveillance using non-negative matrix factorization," *Computational & Mathematical Organization Theory*, vol. 1, no. 11.3, pp. 249–264, 2005.

[5] W Xu, X Liu, and Y Gong, "Document clustering based on non-negative matrix factorization," in *Research and development in informaion retrieval, Proc. 26th Int. ACM SIGIR Conf. on*. 2003, vol. pages, p. 273, ACM.

[6] BØ Wahlgreen and LK Hansen, "Large scale topic modeling made practical," in *Machine Learning for Signal Processing, IEEE Int. Workshop on*. Sept. 2011, vol. 1, pp. 1–6, IEEE.

[7] QW Dong, XL Wang, and L Lin, "Application of latent semantic analysis to protein remote homology detection," *Bioinformatics*, vol. 22, no. 3, pp. 285–290, 2006.

[8] Y Chen and M Kapralov, "Factor modeling for advertisement targeting," *Advances in Neural Information Processing Systems*, 2009.

[9] C. Kemp, J.B. Tenenbaum, T.L. Griffiths, T. Yamada, and N. Ueda, "Learning systems of concepts with an infinite relational model," in *Artificial Intelligence, Proc. National AAAI Conference on*, 2006.

[10] K. Nowicki and T.A.B. Snijders, "Estimation and prediction for stochastic block-structures," *Journal of the American Statistical Association*, vol. 96, no. 455, pp. 1077–1087, 2001.

[11] R.M. Neal, "Markov chain sampling methods for dirichlet process mixture models," *Computational and Graphical Statistics, Journal of*, vol. 9, pp. 249–265, 2000.

[12] TJ Hansen, M Morup, and LK Hansen, "Non-parametric co-clustering of large scale sparse bipartite networks on the GPU," *Machine Learning for Signal Processing, 2011 IEEE International Workshop on*, 2011.

[13] Z. Xu, V.Tresp, S. Yu, K. Yu, and H.-P. Kriegel, "Fast inference in inifinite hidden relational models," *Mining and Learning with Graphs*, 2007.

[14] CM Bishop, *Pattern Recognition and Machine Learning*, Springer, 2007.

[15] G Andrew and J Gao, "Scalable training of L1-regularized log-linear models," *Machine Learning, Proc. 24th Int. Conf. on*, pp. 33–40, 2007.

[16] Y Tsuruoka, J Tsujii, and S Ananiadou, "Stochastic gradient descent training for L1-regularized log-linear models with cumulative penalty," *Annual Meeting of the ACL, Proc. Joint Conf. 47th and Natural Language Processing, 4th Int. Joint Conf. AFNLP*, vol. 1, pp. 477, 2009.

[17] Y Low, J Gonzalez, A Kyrola, D Bickson, C Guestrin, and JM Hellerstein, "Graphlab: A new framework for parallel machine learning," *CoRR*, vol. abs/1006.4990, 2010.

APPENDIX F

Paper 4

Efficient inference of overlapping communities in complex networks

Bjarne Ørum Fruergaard* and Tue Herlau†
Section for Cognitive Systems, DTU Compute, Technical University of Denmark, DK-2800 Kongens Lyngby, Denmark
(Dated: December 1, 2014)

We discuss two views on extending existing methods for complex network modeling which we dub the *communities first* and the *networks first* view, respectively. Inspired by the networks first view that we attribute to White *et al.*[1], we formulate the multiple-networks stochastic blockmodel (MNSBM), which seeks to *separate* the observed network into subnetworks of different types and where the problem of inferring structure in each subnetwork becomes easier. We show how this model is specified in a generative Bayesian framework where parameters can be inferred efficiently using Gibbs sampling. The result is an effective multiple-membership model without the drawbacks of introducing complex definitions of "groups" and how they interact. We demonstrate results on the recovery of planted structure in synthetic networks and show very encouraging results on link prediction performances using multiple-networks models on a number of real-world network data sets.

I. INTRODUCTION

An important theme in modern network science is the inference of structure in complex networks. The simplest and most well studied type of structure is based on partitions of vertices commonly denoted as *blockmodels*. The basic assumption in block modeling is that the vertices are partitioned into non-overlapping sets, the *blocks*, and the probability of observing an edge between two vertices depends only on the block each vertex belong to. This implies vertices in the same block are structurally equivalent [1]. If the partition of vertices into blocks and the other parameters in the model are all considered as random variables in a Bayesian framework the resulting method is commonly known as the *stochastic blockmodel* (SBM) [2].

The SBM has two desirable properties. Firstly, it is sufficiently flexible to capture many different patterns of interaction. Secondly, the model is easy to implement and allows inference of structure in larger networks. When considering extensions of the SBM the following line of reasoning is often followed: The SBM makes use of a latent structure where the vertices are divided into groups or communities. The assumption that everything belongs to exactly one group (friends, family, coworkers, etc.) is too simplistic since these communities often overlap in reality, hence the assumption each vertex belong to one group should be relaxed. This line of thinking lead to two classes of models depending on how the partition-assumption is relaxed. The first type replaces the partition structure of the vertices with a multi-set, that is, a collection of non-empty but potentially overlapping subsets of the vertices [3–5]. The second is a continuous relaxation where for each vertex and each "community" there is a continuous parameter specifying the degree of which the vertex is associated with the community [6–8]. A difficulty with both approaches is that when the assumption of each vertex belonging to a single block is relaxed, the probability that two vertices link to each other must be specified as a function of all the blocks the two vertices are associated with or belong to. The multitude of ingenious ways this problem is solved attests to this being a difficult problem and many of these methods are difficult to implement efficiently and are therefore only applied to small networks.

We wish to emphasize that the above mentioned approaches to extend the basic SBM to models with overlapping blocks, both derive from a more basic assumption; namely that the main goal of block modeling is to model groups or communities of vertices. We dub this view the *communities first* view to emphasize the focus on detecting latent group structure in the network. Comparing to the original work on block modeling by White *et al.*[1], we argue there are two subtle but important distinctions from this more modern interpretation of block modeling: Firstly, that the block only exists as a postulate of structural equivalence between vertices and are specifically not thought to have an interpretation as communities. Secondly, this partitioning of vertices into blocks is only admissible by carefully keeping edges of different *types* as distinct networks. That is to say, that by representing edges of different types as distinct networks, the *simplifying* assumption of stochastic equivalence across blocks becomes permissible. To emphasize this distinction we call this view the *networks first* view.

In this work we propose a method which focuses on the networks first view. We consider a single network as being composed of multiple networks and the principled goal of network modeling is to de-mix this structure into separate networks and model each of the networks with a simpler model. In our work we consider this simplified model to be a stochastic blockmodel, however we emphasize the idea naturally extents to many other types of latent structure including models of overlapping structure. The resulting model, which we name the *multi-network stochastic blockmodel* (MNSBM), has several benefits

i Our sampler for Bayesian inference is easy to specify, the hardest part boiling down to a discrete sampling problem, which can be efficiently parallelized.
ii The inference is nonparametric in that it infers the model order automatically for each subnetwork.
iii The method is easily extended to include hybrid models such as models of overlapping hierarchies and blockmodels.

The remainder of the paper is organized as follows: In sec-

* bowa@dtu.dk; Also at Adform ApS, DK-1103 København K, Denmark
† tuhe@dtu.dk

tion I A we will argue more carefully that the networks first view is found in the original work by White *et al.*, in section II we will introduce the MNSBM and in section III demonstrate our model is able to de-mix planted structure in synthetic data as well as successfully increase link prediction performance on real-world data sets by modeling networks as multiple, overlapping SBMs.

A. Assumptions of block modeling

White *et al.* considers models for multiple networks defined on the same set of vertices. An example is the Samson monastery networks dataset [9] in which there are 8 networks where each network is comprised by the answer of the monks to specific question such as the degree to which they like, praise, esteem, etc., other monks. In the terminology of the article each of these networks represents a *type* and an edge in a particular network is an edge of that type. For instance an edge between two monks can be of the type "like" or "praise" etc. The distinction into multiple networks is taken as fundamental:

> We take as given the incidence of each of several distinct types of tie (...). Each is a separate network to be contrasted with other such networks, rather than merged with them to form a complex bond between each pair of actors. This analytic segregation of network types is basic to our framework [1, p731]

It is worth emphasizing why according to White *et al.* the segregation into different networks is considered basic. The blockmodel hypothesis is given as five points, and we pay special attention to the following two:

> First, structural equivalence requires that members of the population be partitioned into distinct sets, each treated homogeneously not only in its internal relations but also in its relations to each other such set. (...) Third, many different types of tie [edges] are needed to portray the social structure [i.e. communities] of a population.[1, p739]

However if a block is simply defined as structurally equivalent vertices (as opposed to a community of vertices), it is on this definition no longer obvious what it means for two blocks to overlap since the overlap would break structural equivalence. The de-emphasis on blocks as capturing explicit group structure and emphasis on the need for types of ties lead us to dub this the *networks first view*.

It is worth contrasting this with a modern view on blockmodels where blocks are taken to signify structure. For instance Latouche *et al.* [10] first discuss the blockmodel as introduced by White *et al.* and then discuss the point of contention:

> A drawback of existing graph clustering techniques is that they all partition the vertices into disjoint clusters, while lots of objects in real world applications typically belong to multiple groups or communities. [10, p310]

thus a block or group in this view clearly reflects a real entity thought to exist in the graph, which vertices may or may not belong to, and not simply structural equivalence; hence the term *communities first view*. We wish to emphasize that we do not consider the communities first view on network modeling as being wrong or mistaken, for instance Latouche *et al.* (and references therein) consider many concrete instances where it is thought to hold. However, we consider it a particular hypothesis on the structure of the network composed of the following two assumptions; (i) the network is composed of links of homogeneous type, and (ii) the network should be thought of as containing groups the vertices may be members to and these groups explain the links. On the contrary, the networks first view considers the complexity of the network as primarily being derived from it containing a mixture of networks of many types. For instance when we collect a network of friendships on an online social network, we may suppose the friendships fall into several categories such as "friendship"-ties, "family"-ties, "work colleague"-ties, and so on. It is the overlap of these distinct types of ties that induces much of the difficulty in modeling complex networks, and when these networks are kept separate, the problem of inferring structure simplifies potentially to the point where naive assumptions such as structural equivalence (as in the SBM) may suffice.

II. METHODS

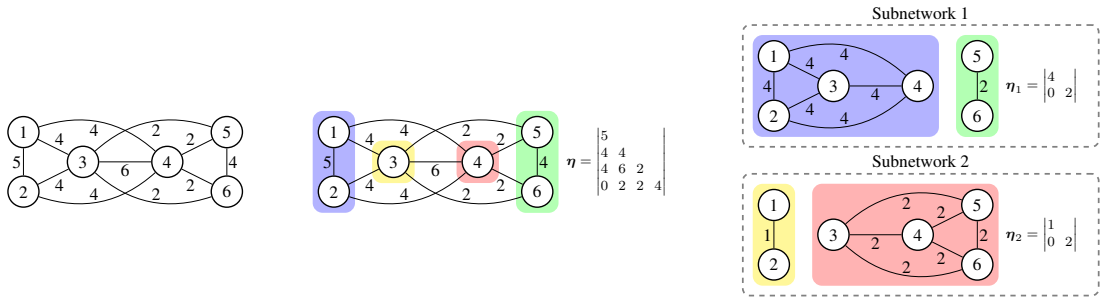
Consider a network comprised of n vertices, $1, 2, \ldots, n$, and let A^*_{ij} denote the observed number of edges between vertices i and j. The reason we allow multiple edges between vertices will be apparent later, however for simplicity (but without loss of generality) we will otherwise consider $\boldsymbol{A}^*$ as being symmetric and without self-loops, i.e. $A^*_{ij} = A^*_{ji}$ and $A^*_{ii} = 0$. In line with the networks first view we consider $\boldsymbol{A}^*$ as arising from multiple networks, $\boldsymbol{A}^1, \ldots, \boldsymbol{A}^S$, which have been aggregated due to an unknown data registration process. In the terminology above each network is comprised by a particular type of edge corresponding to for instance different social relations. We will denote the process of aggregation by a function h. In principle this could be a function of S networks, however we will make the assumption

$$A^*_{ij} = h(A^1_{ij} + A^2_{ij} + \cdots + A^S_{ij}) \tag{1}$$

and in particular be interested in the case where h is the heavyside step function H, defined as 1 when the input is positive and otherwise zero. This corresponds to the natural assumption; we discover an edge between two vertices if there is an edge in any of the networks of different edge types. Next we assume each network $\boldsymbol{A}^s$ arise from a model $\mathcal{M}_s$ with latent parameters Φ_s. In this case

$$\Phi^1, \ldots, \Phi^S \sim P(\cdot) \qquad \textit{not necessarily independently} \tag{2}$$

$$A^s_{ij} \sim P(\cdot|\Phi^s) \quad \textit{independently} \tag{3}$$



(a) Observed network (b) MNSBM, $S = 1$ (c) MNSBM, $S = 2$

FIG. 1. (Color online) Examples of networks illustrating the generative models of MNSBM. (a) An example of an undirected multigraph where edge weights represent the number of edges. (b) The same network with a clustering into 4 vertex communities according to MNSBM with a single network, which on average generates the observed network. (c) The same network again, but this time divided in two as per an MNSBM of two networks. The summation of the networks in (c) will also on average generate the observed edges.

$$p(\boldsymbol{A}^*, \boldsymbol{A}^1, \dots, \boldsymbol{A}^S, \Phi^1, \dots, \Phi^S) \tag{4}$$

$$= p(\Phi^1, \dots, \Phi^S) \prod_{i<j} \delta_{A^*_{ij} - h(\sum_s A^s_{ij})} \prod_s p(A^s_{ij}|\Phi^s) \tag{5}$$

From this we can easily extract the marginal probabilities:

$$p(A^s_{ij}|\cdots) \propto p(A^s_{ij}|\Phi^s)\delta_{A^*_{ij} - h(\sum_s A^s_{ij})} \tag{6}$$

$$p(\Phi^s|\cdots) \propto p(A^s|\Phi^s)p(\Phi^1, \dots, \Phi^S) \tag{7}$$

An important special case is if the marginal probabilities $p(A^s_{ij}|\Phi^s)$ are Poisson distributed and the parameters $\Phi^1, \dots, \Phi^S$ are independent. We may write this as $p(A^s_{ij}|\Phi^s) = \mathcal{P}(A^s_{ij}|\eta^s_{ij})$ where $\mathcal{P}$ denotes the Poisson distribution and the rates η^s_{ij} are considered part of the parameter vector Φ^s. Recall the following basic properties of Poisson random variables: If $X_1, \dots, X_k$ is a set of k independent Poisson random variables then their sum $X = \sum_i X_i$ is Poisson distributed with rate $\eta = \sum \eta_i$ and the conditional distribution of $(X_1, \dots, X_k)$ on $X = k$ is distributed as a multinomial distribution ($\mathcal{M}$):

$$X_1, \dots, X_k | X = n \sim \mathcal{M}\left(\cdot\,|\frac{\eta_1}{\eta_0}, \cdots, \frac{\eta_k}{\eta_0}, n\right) \tag{8}$$

Introducing $\eta_{ij} = \sum_{s=1}^S \eta^s_{ij}$ then with these assumptions it follows

$$p(A_{ij}|\cdots) \propto \mathcal{P}(A_{ij}|\eta_{ij})\delta_{(A^*_{ij} - h(A_{ij}))} \tag{9}$$

$$p(A^1_{ij}, \dots, A^S_{ij}|\cdots) \propto \mathcal{M}\left(A^1_{ij}, \dots, A^S_{ij}|\frac{\eta^1_{ij}}{\eta_{ij}}, \cdots, \frac{\eta^S_{ij}}{\eta_{ij}}, A_{ij}\right) \tag{10}$$

$$p(\Phi^s|\cdots) \propto p(A^s|\Phi^s)p(\Phi^s) \tag{11}$$

Since the restriction in eq. (9) is on a univariate density it will for all reasonable choices of h be easy to handle analytically. In the particular case of the Heaviside function we have

$$p(A_{ij}|\cdots) = \begin{cases} \delta_{A_{ij}} & \text{if } A^*_{ij} = 0, \\ \frac{\mathcal{P}(A_{ij}|\eta_{ij})}{1-\mathcal{P}(0|\eta_{ij})} & \text{otherwise.} \end{cases} \tag{12}$$

Accordingly, eqs. (9) and (10) may be sampled very quickly independently for each edge in the observed network $\boldsymbol{A}^*$ and sampling eq. (11) is only as complex as sampling a single network model and, when considering S networks, these parameters may be sampled independently of each other.

A. Overlapping networks for the stochastic blockmodels

Under the above assumptions of Poisson observations any model for single networks that can be re-formulated to have Poisson observations can be used in a multi-network setting. This include diverse types of network models including for instance the hierarchical network model of Clauset *et al.* [11] or the overlapping community-type models such as [3, 10] which are easy to re-formulate as Poisson observations (trivially if one simple consider the Bernoulli probability as the rate in the Poisson model) or in the case of [12] or [7] already formulated in terms of Poisson observations, and one can also consider hybrids where different mixtures of models are used.

However we will consider the simplest case where each network is modeled as a SBM. Many popular references exist on the SBM [2, 13, 14] however we will re-state it for completeness. The SBM assumes the n vertices are divided into ℓ non-overlapping blocks. The assignment of vertex i to block ℓ in network s is indicated by $z^s_i = \ell$ and we denote by $\boldsymbol{z}^s = (z^s_1, \dots, z^s_n)$ the assignment vector of all vertices in network s. As a prior for assignments we use the Chinese Restaurant Process (CRP) parameterized by a single parameter α controlling the distribution of group size [15], which we indicate by the symbol $\mathcal{C}$. Using our notation, this has a density given as

$$p(\boldsymbol{z}^s|\alpha) = \frac{\alpha^L \Gamma(\alpha)}{\Gamma(N+\alpha)} \prod_{\ell=1}^L \Gamma(n_\ell), \tag{13}$$

where $\Gamma(\cdot)$ is the Gamma function and $\ell = 1, \dots, L$ indexes the blocks of network s. For further details on why the CRP

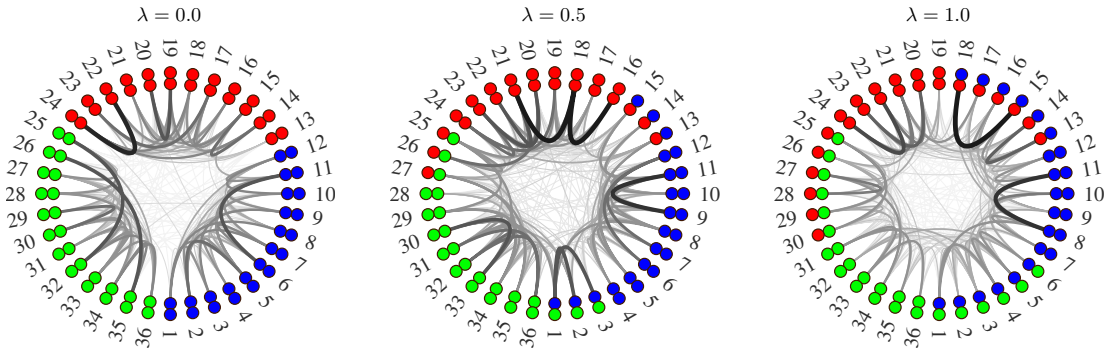


FIG. 2. (Color online) Three examples of networks sampled from a synthetic model with $N = 36$ and $K_1 = K_2 = 3$ are shown with $\lambda = \{0.0, 0.5, 1.0\}$, respectively. Two subnetworks are represented in an inner and outer ring of nodes with colors representing cluster assignments in each subnetwork. The darkness and width of edges represent edge weights.

is advantageous over, say a uniform prior, see [16, p.2]. The generative process we then write up as:

$$\boldsymbol{z}^s \sim \mathcal{C}(\alpha), \qquad \textit{clusters} \tag{14a}$$

$$\text{for } \ell \leq m \qquad \eta^s_{\ell m} \sim \mathcal{G}(\kappa, \lambda), \qquad \textit{link rate} \tag{14b}$$

$$\text{for } i < j \qquad A^s_{ij} | \boldsymbol{z}^s, \boldsymbol{\eta}^s \sim \mathcal{P}(\eta_{z_i z_j}), \qquad \textit{link weight} \tag{14c}$$

In words, this process can be understood as follows

(i) $\boldsymbol{z}^s \sim \mathcal{C}(\alpha)$: Sample cluster assignments from the Chinese Restaurant Process [15] parametrized by a single parameter α controlling the distribution of group size, thus obtaining the partitioning of cluster associations for each vertex ($|\boldsymbol{z}^s| = N$) into $1 \leq L \leq N$ clusters.

(ii) $\eta^s_{\ell m} \sim \mathcal{G}(\kappa, \lambda)$: Generate intra- and intercluster link rates from a Gamma distribution with shape parameter κ and rate parameter λ.

(iii) $A^s_{ij} | \boldsymbol{z}^s, \boldsymbol{\eta}^s \sim \mathcal{P}(\eta_{z_i z_j})$: Generate edges that are independently Poisson distributed with the expected number of links between vertices i and j being $\eta_{z_i z_j}$.

An illustrative example of single-network and two-network MNSBM is shown in FIG. 1. Here the shaded regions in (b) and (c) surrounding the vertices represent blocks and next to the networks are given corresponding link rates. Both models FIG. 1b and FIG. 1c will, given Poisson observation models, on average generate the observed network FIG. 1a.

B. Inference and missing data

An efficient inference scheme is easily obtained through eqs. (9), (10) and (11). Notice when updating Φ^s in eq. (11) one can use the standard tool to integrate out the $\boldsymbol{\eta}^s$ parameters. This leaves only the assignments $\boldsymbol{z}^s$ and hyperparameters α^s, κ^s, λ^s to be sampled. We sampled $\boldsymbol{z}^s$ using standard Gibbs sampling and the hyperparameters using random-walk Metropolis-Hastings in log-transformed coordinates [14]. For simplicity we assumed a $\mathcal{G}(2, 1)$ prior for α^s, κ^s, λ^s. For details on deriving the collapsed Gibbs sampler, we refer to [16].

To predict missing edges we used imputation. Suppose an edge ij is unobserved. Then if we implement the sampler exactly as described but for this pair ij replace eq. (9) with the unconstrained distribution

$$p(A_{ij} | \cdots) = \mathcal{P}(A_{ij} | \eta_{ij}) \tag{15}$$

we will get a sequence of MCMC estimates of A_{ij}, $(a^{(t)}_{ij})^T_{t=1}$. Predictions may then be estimated as the MCMC average

$$p(A_{ij} = 1 | \boldsymbol{A}^*) = \frac{1}{T} \sum_{t=1}^{T} h(a^{(t)}_{ij}). \tag{16}$$

In terms of computational complexity, a single Gibbs sweep over $\boldsymbol{z}^s$ scales as $\mathcal{O}(EK^2)$, where $E = \sum_{ij} \boldsymbol{A}^s_{ij}$, is the number of realized edges in $\boldsymbol{A}^s$ (notice this is lower than the number of observed edges), and K is the number of components. In addition the multinomial re-sampling steps in eq. (9),(10) scales as $\mathcal{O}(E)$ taking up only a small fraction of the time spent. Thus with computational complexity $\mathcal{O}(EK^2)$ per Gibbs sweep. While this would indicate an computational cost roughly S times greater than a single SBM it is worth emphasizing the stochastic de-coupling of networks in eq. (11) admits a very easy parallelization over Gibbs sweeps and with the advent of multi-core machines this allows the method to parallelize easily which we made use of. In addition in section III B we will show the use of multiple networks allow each network to be modeled using fewer blocks reducing the quadratic factor in the cost. Taken together the cost of our method, when S was lower than the number of cores on the machine, was comparable to the $S = 1$ case.

III. RESULTS AND DISCUSSION

We test our method on both synthetic (computer generated) networks as well as a number of real world network datasets. Our synthetic benchmarks allow us to test the sampler as well

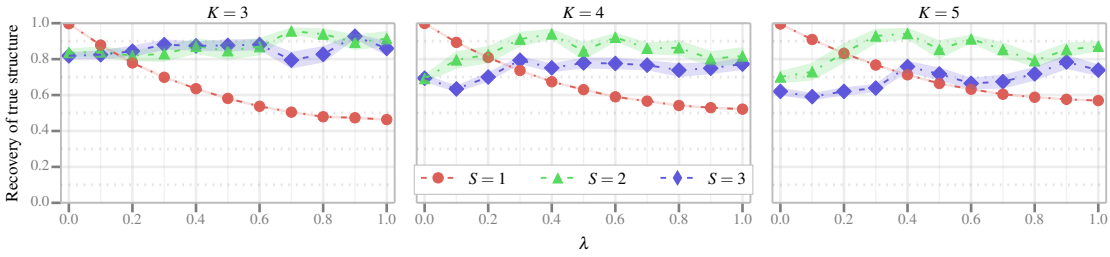


FIG. 3. (Color online) The average AUC scores for each of $K = 3, 4, 5$ and for $\hat{S} = 1, 2, 3$ with varying λ for synthetic networks and ground truth being $S = 2$. Each point is the average over 20 random restarts and shaded regions represent standard deviations of the mean. The experiments were run for $T = 3000$ iterations and the AUC for each experiment is computed from the estimated same-block probabilities averaged over all Markov samples in the last 500 iterations.

as monitor how well the model identifies ground truth, i.e., planted structure, under controlled conditions. Analyzing real networks gives us an idea of the performance of our model in real-world scenarios.

A. Synthetic networks

Since MNSBM is a generative model of networks, we can artificially sample data from known parameters. Running the model and comparing the solutions with ground truth, allows us to monitor how well the model performs when conditions are varied, e.g., community structure and area of overlap.

1. *Generating synthetic networks*

The synthetic data that we generate will serve to demonstrate that MNSBM can recover solutions with structure close to the ground truth.

We generate synthetic data by sampling from an instantiated MNSBM model with two subnetworks. In each subnetwork, we put $K_1 = K_2 = K$ equally sized clusters, i.e., the same number of communities for each subnetwork. We set the diagonal of $\boldsymbol{\eta}^1$ to 1, the diagonal of $\boldsymbol{\eta}^2$ to 1.5 and the off-diagonals of both to 0.1 everywhere, thus generating strong community structure in both subnetworks. Setting slightly different link rates in the diagonals of $\boldsymbol{\eta}^1$ and $\boldsymbol{\eta}^2$ helps with the identifiability of the true structure. The overlap in the generated networks we control by circular shifting the clusters in the second subnetwork. Hence, in a network of N nodes and a circular shift of m, the number of overlapping nodes is mN/K, where for notational simplicity we assume N is a multiple of K. Consequently, with a shift $m > 0$, there are $N - mN/K$ nodes in non-overlapping clusters as well as mN/K nodes in overlapping clusters. Since we control overlap by a circular shift of the clusters in one subnetwork, the structure is trivially symmetric around $m = N/(2K)$, i.e., structurally there is no difference whether we shift $m = N/(2K) - i$ or $m = N/(2K) + i$ for all $i \leq N/(2K)$. Therefore we are interested in varying the overlap $m = 0, 1, \ldots, N/(2K)$ and we define a parameter $\lambda = 2Km/N$ as a discrete scale between 0 and 1, measuring from no shift up to the maximum $N/(2K)$. In FIG. 2 we show an example of synthetic networks generated as described above using $N = 30$, $K = 3$, and $\lambda = \{0.2, 0.6, 1.0\}$.

Using $K = \{3, 4, 5\}$, we generate networks with $N = \{60, 80, 100\}$ nodes, respectively, and vary λ in the interval $(0, 1)$.

2. *Performance on synthetic networks*

When testing our model on synthetic data, we are interested in how well our inference procedure is able to identify planted structure. In order to measure similarity between true and estimated models, however, we identify that similarity measures on the true and estimated assignment vectors directly introduces a matching problem. To circumvent this problem, we opt instead for a measure of match between the overall structures as follows. Suppose we are trying to infer the block structure of an artificially constructed graph, $\boldsymbol{A}^*$, containing S^* true subnetworks using a MNSBM with $S > S^*$ subnetworks. Then, assuming the MNSBM works correctly, one of the subnetworks will be empty and the community structure will not be informative. To avoid empty subnetworks to influence our results, we will therefore focus on whether the MNSBM partitions the realized edges correctly. This can be done by, for each edge $e_k = (i, j)$ of $\boldsymbol{A}^*$, determine (i) if the particular true subnetwork which generated $\boldsymbol{A}^*$, $\boldsymbol{A}^{*s}$, assigned (i, j) to the same block and (ii) compare this to whether the subnetwork A^s in the inferred structure which "explains" (i, j) (i.e. has $A^s_{ij} = 1$) also assigns (i, j) to the same block. Since each true edge may be explained by multiple subnetworks, we simply compute the weighted average over each subnetwork explaining this edge. This results in a binary vector of a_k's consisting of the true same-block information and a weighted vector of w_k's consisting of the estimated same-block probability averaged over all T Markov samples. Specifically, for any given edge $e_k = (i, j)$ we de-

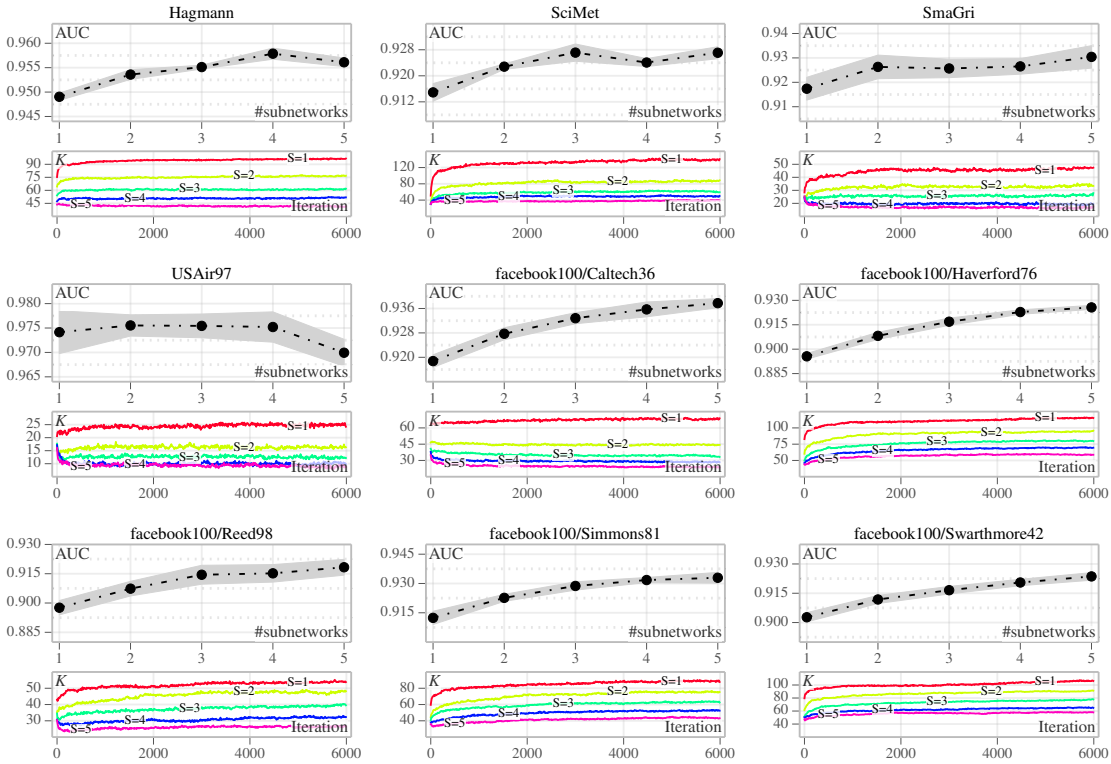


FIG. 4. (Color online) MNSBM results for each of the networks introduced in the text. For each network, MNSBM is run with $S = \{1, 2, 3, 4, 5\}$, i.e., varying number of subnetworks, and for 6,000 iterations. In the upper plots the average AUC link prediction scores are shown as a function of S. The AUC score for a single chain is computed as the average of the predictions from every 10^{th} iteration of the last half of the chain, discarding the first 3,000 as burn-in. The averages are based on 5 random restarts and 5% edges and non-edges picked randomly for testing. Shaded regions represent the standard deviation of the mean. In the lower plots we show the average number of detected components per subnetwork as a function of iterations, with each line representing an MNSBM with a different S. The averages are computed similarly to the AUC scores.

fine:

$$a_k = \sum_{s=1}^{S} A_{ij}^{*s} \delta_{z_i^{*s} = z_j^{*s}}, \tag{17}$$

$$w_k = \frac{1}{\sum_s^S A_{ij}^s} \sum_{s=1}^{S} A_{ij}^s \delta_{z_i^s = z_j^s} \tag{18}$$

where $\delta_{(\cdot)}$ is an indicator function that is 1 if the condition $(\cdot)$ is true and 0 otherwise. We can then compute area under the curve (AUC) of the receiver operating characteristics based on a_k and w_k.

In FIG. 3 we show the dependencies of K (plot titles), the choice of $\hat{S}$ (different curves) and the overlap parameter λ. We observe that for $\lambda = 0$, the simplest network model is always best, which is unsurprising since there is no overlapping structure. Since the single network, $\hat{S} = 1$, infers disjoint clusters for the overlapping structure, which increases in size as $\lambda \to 1$, we see a decline in similarity. For $\lambda > 0.2$ we begin to see the model with two subnetworks, $\hat{S} = 2$, outperforming the single network model. With the networks sizes that we have chosen for these experiments, $\lambda = 0.1$ means there is only one node in each overlap and it looks as if it is necessary with a bit more nodes per overlap, i.e., three ($\lambda = 0.3$), in order for the $\hat{S} = 2$ model to consistently pick up the structure.

For $\hat{S} = 3$, where the model allows too many subnetworks, we see the similarity degrades compared to using $\hat{S} = 2$. I.e., the sampler is not perfectly able to set to zero all the edges of one of the subnetworks. As we will see in our experiments on real networks, choosing $\hat{S}$ too large does however not necessarily mean worse predictive performance. In practice one seldom knows the underlying structure, so for empirically assessing which $\hat{S}$ is better, we must resort to other strategies. In that case, we suggest cross-validating $\hat{S}$ on a held-out sample of edges w.r.t. a problem specific measure such as AUC, RMSE, MAE, etc., depending on the task at hand, and picking

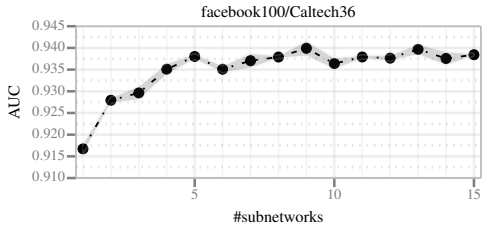


FIG. 5. AUC score as a function of the number of subnetworks for the *facebook100/Caltech36* network. The plotted values are averages and the shaded region represents standard deviation of the mean based on the same experimental settings as in FIG. 4.

$\hat{S}_{\rm opt}$ based on a plot of the target measure as a function of $\hat{S}$.

B. Real networks

We have tested MNSBM on a number of real world network datasets, where we are interested in *link prediction* for out-of-sample edges. We vary the number of sub-networks S from 1 to 5 and repeatedly run inferences five times with random initialization and a random subset of 5% edges (and a similar number of non-edges) held-out for measuring link prediction performance. The link prediction performance we report in area under curve (AUC) of the receiver operating characteristics.

The nine networks we analyze are.

(i) *Hagmann:* undirected weighted network of the number of links between 998 brain regions as estimated by tractography from diffusion spectrum imaging across five subjects [17]. As in [16], the graph of each subject has been symmetrized, thresholded at zero and the five subject graphs added together.

(ii) *SciMet:* directed weighted network of a citation network between 3,086 authors in the Scientometrics journal 1978-2000 [18].

(iii) *SmaGri:* directed weighted network of another citation network between 1,059 authors to Small & Griffith and Descendants [18].

(iv) *USAir97:* undirected weighted network of air traffic flow between 332 U.S. airports in 1997 [18–20].

(v) *facebook100/*:* undirected unweighted networks from five friendship networks in U.S. colleges from the Facebook100 dataset [21]. *Caltech36:* 769 nodes, *Haverford76:* 1446 nodes, *Reed98:* 962 nodes, *Simmons81:* 1518 nodes, *Swarthmore42:* 1659 nodes.

For all the directed networks, we symmetrize them, i.e., treat them as undirected, and for the weighted networks, we make them unweighted by treating any non-zero edge as a link.

The results are shown in FIG. 4. For each of the networks, we show in the top the average AUC scores with the shaded region being standard deviations of the average based on five random restarts and different held-out samples. In the bottom, we show the evolution of average number of inferred communities per subnetwork. As we increase S, the number of subnetworks in MNSBM, we generally see the AUC scores increasing, meaning that the link prediction for missing edges improves. The only exception is *USAir97*, which is also the smallest network in our benchmark. Either *USAir97* does not exhibit overlapping structure or the small size of the dataset is an inhibiting factor. In terms of average number of inferred communities, we see that this is consistently decreasing as we increase S and eventually saturates. These experiments confirm that on a variety of real world networks, MNSBM enables modeling ensembles of simpler substructures while increasing link prediction performance.

As a separate experiment, we have run additional five independent trials with the *facebook100/Caltech36* network, where we let S increase to 15. The AUC as a function of S is shown in FIG. 5. We see that after $S = 5$ the performance saturates, which shows that our method is robust towards choosing too high number of subnetworks. I.e., overfitting is not really a concern here. In practice what happens is that as the sampler progresses, superfluous subnetworks get very few edges assigned (if any), hence they become redundant without affecting the performance negatively.

IV. CONCLUSION

In this work we have discussed two views on extending existing methods for structural network modeling. In the *communities first* view, which we argue is prominent in recent complex networks research, the (simplifying) assumption of structural equivalence is sacrificed in order to allow for overlapping groups leading to evermore complicated definitions on what constitutes a "group" and how these groups interact. We explore an alternative view which we attribute to the seminal work of White *et al.*, which we dub the *networks first* view. The key distinction is that it considers the complexity of observed networks as arising as a consequence of multiple networks of different types of ties (edges) being aggregated. Inspired by the latter view of complex networks we introduce the multiple-networks stochastic blockmodel (MNSBM), which seeks to *separate* the observed network into subnetworks of different types and where the problem of inferring structure in each subnetwork can benefit from the simplifying assumption of structural equivalence. The result is effectively the joint inference problem of splitting the observed edges between subnetworks and identifying (block) structure in each subnetwork. We formulate this model in a generative Bayesian framework over parameters that can be inferred efficiently using Gibbs sampling. Thereby we obtain an effective multiple-membership model without introducing the drawbacks that originate from defining complex interactions between groups. We demonstrate results on the recovery of planted structure in synthetic networks, as well as provide results in terms of link prediction performances on a number of real-world network data sets, which highly motivate the use of multiple subnetworks over a naive stochastic blockmodel, assuming disjoint blocks globally on the networks.

[1] Harrison C White, Scott A Boorman, and Ronald L Breiger, "Social structure from multiple networks. I. Blockmodels of roles and positions," American journal of sociology , 730–780 (1976).

[2] Paul W Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt, "Stochastic blockmodels: First steps," Social networks **5**, 109–137 (1983).

[3] Kurt T Miller, Thomas L Griffiths, and Michael I Jordan, "Nonparametric latent feature models for link prediction." in *NIPS*, Vol. 9 (2009) pp. 1276–1284.

[4] K Palla, D Knowles, and Z Ghahramani, "An Infinite Latent Attribute Model for Network Data," in *International Conference on Machine Learning* (2012).

[5] Morten Morup, Mikkel N Schmidt, and Lars Kai Hansen, "Infinite multiple membership relational modeling for complex networks," in *Machine Learning for Signal Processing (MLSP), 2011 IEEE International Workshop on* (IEEE, 2011) pp. 1–6.

[6] Edoardo M Airoldi, David M Blei, Stephen E Fienberg, and Eric P Xing, "Mixed membership stochastic blockmodels." Journal of Machine Learning Research **9**, 3 (2008).

[7] Brian Ball, Brian Karrer, and M. E. J. Newman, "Efficient and principled method for detecting communities in networks," Phys. Rev. E **84**, 036103 (2011).

[8] Aditya Krishna Menon and Charles Elkan, "Link prediction via matrix factorization," in *Machine Learning and Knowledge Discovery in Databases* (Springer, 2011) pp. 437–452.

[9] Samuel F Sampson, *Crisis in a cloister*, Ph.D. thesis, Ph. D. Thesis. Cornell University, Ithaca (1969).

[10] Pierre Latouche, Etienne Birmelé, and Christophe Ambroise, "Overlapping stochastic block models with application to the French political blogosphere," The Annals of Applied Statistics **5**, 309–336 (2011).

[11] Aaron Clauset, Cristopher Moore, and Mark EJ Newman, "Hierarchical structure and the prediction of missing links in networks," Nature **453**, 98–101 (2008).

[12] Aditya Krishna Menon and Charles Elkan, "Predicting labels for dyadic data," Data Mining and Knowledge Discovery **21**, 327–343 (2010).

[13] Michelle Girvan and Mark EJ Newman, "Community structure in social and biological networks," Proceedings of the National Academy of Sciences **99**, 7821–7826 (2002).

[14] Charles Kemp, Joshua B Tenenbaum, Thomas L Griffiths, Takeshi Yamada, and Naonori Ueda, "Learning systems of concepts with an infinite relational model," in *AAAI*, Vol. 3 (2006) p. 5.

[15] David Aldous, "Exchangeability and related topics," École d'Été de Probabilités de Saint-Flour XIII—1983 , 1–198 (1985).

[16] Tue Herlau, Mikkel N. Schmidt, and Morten Mørup, "Infinite-degree-corrected stochastic block model," Phys. Rev. E **90**, 032819 (2014).

[17] Patric Hagmann, Leila Cammoun, Xavier Gigandet, Reto Meuli, Christopher J Honey, Van J Wedeen, and Olaf Sporns, "Mapping the structural core of human cerebral cortex," PLoS biology **6**, e159 (2008).

[18] V Batagelj and A Mrvar, "Pajek datasets," http://vlado.fmf.uni-lj.si/pub/networks/data/ (2006).

[19] Zachary Neal, "Refining the air traffic approach to city networks," Urban Studies **47**, 2195–2215 (2010).

[20] Zachary Neal, "AIRNET: A Programme for Generating Intercity Networks," Urban Studies **51**, 136–152 (2014).

[21] Amanda L. Traud, Peter J. Mucha, and Mason A. Porter, "Social Structure of Facebook Networks," (2011), arXiv:1102.2166.

Nomenclature

AUC Area under the curve of the receiver operating characteristic, page 28

CPU Central processing unit, the general-purpose processor in a computer, page 50

GPU Graphics processing unit, a massively parallel processor specialized for visualizing graphics in real-time, i.e., for computer games, page 47

CPC_t Target cost per click, a target specified by an advertiser of how much they are willing to pay per click, page 4

CF Collaborative filtering, a technique often at the core of recommendation engines, page 35

CF Collaborative filtering, page 37

CPC Cost per click, an indirect measure of the value of showing and advertisement, page 4

DSP Demand side platform, a platform enabling agencies to buy advertising space through real-time bidding auctions on online ad exchanges, page 3

GPGPU General purpose GPU computing, page 50

iid. Independent and identically distributed, page 11

$L(\cdot)$ A loss function, page 13

LL Logistic loss, page 28

MAP Short for *maximum posteriori* estimation, page 11

MCMC Markov Chain Monte Carlo, a sampling technique, page 23

MLE Maximum likelihood estimation/estimator, page 10

$\Omega(\cdot)$ A regularization function or simply *regularizer*. Also called a *penalization* term, page 13

$\mathbb{P}\mathrm{r}$ Probability distribution function, page 10

pdf Probability density function, page 8

RTB Real-time bidding, page 3

SGD Stochastic gradient descent, or *on-line learning*, is a method for solving large-scale optimization problems based on approximations to the true gradient, page 21

SIMD Single instruction multiple data, a parallel processing execution model, page 52

SMP Symmetric multiprocessor, a single core on a graphics processing unit., page 50

References

[1] Airoldi, E. and Blei, D. Mixed Membership Stochastic Blockmodels. *Journal of Machine Learning Research*, 9:1981–2014, 2008.

[2] Aldous, D. J. Exchangeability and related topics. *École d'Été de Probabilités de Saint-Flour XIII — 1983*, pages 1–198, 1983. ISSN 10477047. doi: 10.1007/BFb0099421.

[3] Allesiardo, R., Féraud, R., and Bouneffouf, D. A Neural Networks Committee for the Contextual Bandit Problem. In Loo, C. K., Yap, K. S., Wong, K. W., Teoh, A., and Huang, K., editors, *Neural Information Processing*, volume 8834 of *Lecture Notes in Computer Science*, pages 374–381. Springer International Publishing, Cham, 2014. ISBN 978-3-319-12636-4. doi: 10.1007/978-3-319-12637-1.

[4] Andrew, G. and Gao, J. Scalable training of L1-regularized log-linear models. *Proceedings of the 24th international conference on Machine learning*, pages 33–40, 2007.

[5] Ball, B., Karrer, B., and Newman, M. E. J. Efficient and principled method for detecting communities in networks. *Physical Review E*, 84(3):036103, September 2011. ISSN 1539-3755. doi: 10.1103/PhysRevE.84.036103.

[6] Bamber, D. The area above the ordinal dominance graph and the area below the receiver operating characteristic graph. *Journal of Mathematical Psychology*, 12 (4):387–415, November 1975. ISSN 00222496. doi: 10.1016/0022-2496(75)90001-2.

[7] Barto, A. *Reinforcement learning: An introduction*. MIT Press, 1998.

[8] Bennett, J. and Lanning, S. The netflix prize. *In KDD Cup and Workshop in conjunction with KDD*, 2007.

[9] Berry, D. and Fristedt, B. *Bandit Problems: Sequential Allocation of Experiments (Monographs on Statistics and Applied Probability)*. Chapman & Hall, London, 1985.

[10] Berry, M. and Browne, M. Email surveillance using non-negative matrix factorization. *Computational & Mathematical Organization Theory*, 1(11.3):249–264, 2005.

[11] Bishop, C. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 2007. ISBN 0387310738.

[12] Blei, D. and Ng, A. Latent dirichlet allocation. *The Journal of Machine Learning*, 3:993–1022, 2003.

[13] Bottou, L. and Peters, J. Counterfactual reasoning and learning systems: the example of computational advertising. *The Journal of Machine …*, 2013.

[14] Bouneffouf, D., Bouzeghoub, A., and Gançarski, A. A contextual-bandit algorithm for mobile context-aware recommender system. In *Neural Information Processing*, volume 7665 of *Lecture Notes in Computer Science*, pages 324–331. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-34486-2. doi: 10.1007/978-3-642-34487-9.

[15] Broder, A. and Josifovski, V. Lecture Introduction to Computational Advertising. *Stanford University, Computer Science. Online lecture notes.*, 2009.

[16] Chen, Y., Pavlov, D., and Canny, J. Large-scale behavioral targeting. *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2009.

[17] Davis, J. and Goadrich, M. The relationship between Precision-Recall and ROC curves. In *Proceedings of the 23rd international conference on Machine learning - ICML '06*, pages 233–240, New York, New York, USA, 2006. ACM Press. ISBN 1595933832. doi: 10.1145/1143844.1143874.

[18] Dhillon, I. S. Co-clustering documents and words using bipartite spectral graph partitioning. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '01*, pages 269–274, New York, New York, USA, 2001. ACM Press. ISBN 158113391X. doi: 10.1145/502512.502550.

[19] Dhillon, I., Guan, Y., and Kulis, B. Kernel k-means: spectral clustering and normalized cuts. *In Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 551–556, 2004.

[20] Dhillon, I., Guan, Y., and Kulis, B. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(11):1944–1957, 2007.

[21] Dong, Q., Wang, X., and Lin, L. Application of latent semantic analysis to protein remote homology detection. *Bioinformatics*, 22(3):285–290, 2006.

[22] Easley, D. and Kleinberg, J. *Networks, Crowds, and Markets*. Cambridge University Press, Cambridge, 2010. ISBN 9780511761942. doi: 10.1017/CBO9780511761942.

[23] Efron, B. Bootstrap methods: another look at the jackknife. *The annals of Statistics*, 1979.

[24] Elmer, G. *Profiling Machines: Mapping the Personal Information Economy*. MIT Press, 2003. ISBN 0262262584.

[25] Fawcett, T. An introduction to ROC analysis. *Pattern Recognition Letters*, 27 (8):861–874, June 2006. ISSN 01678655. doi: 10.1016/j.patrec.2005.10.010.

[26] Fruergaard, B. O. Predicting clicks in online display advertising with latent features and side-information. *ArXiv e-prints arXiv:1411.7924 [stat.ML]*, 2014.

[27] Fruergaard, B. O. and Hansen, L. K. Compact web browsing profiles for click-through rate prediction. In *Machine Learning for Signal Processing (MLSP), 2014 IEEE International Workshop on*, pages 1–6, September 2014. doi: 10.1109/MLSP.2014.6958852.

[28] Fruergaard, B. O. and Herlau, T. Efficient inference of overlapping communities in complex networks. *ArXiv e-prints arXiv:1411.7864 [stat.ML]*, 2014.

[29] Fruergaard, B. O., Hansen, T. J., and Hansen, L. K. Dimensionality reduction for click-through rate prediction: Dense versus sparse representation. *ArXiv e-prints arXiv:1311.6976 [stat.ML]*, November 2013.

[30] Geman, S. and Geman, D. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *Pattern Analysis and Machine Intellligence, IEEE Transactions on*, 6:721—-741, 1984.

[31] Gittins, J., Glazebrook, K., and Weber, R. *Multi-armed Bandit Allocation Indices*. John Wiley & Sons, Ltd, 1989.

[32] Golub, G. and Reinsch, C. Singular value decomposition and least squares solutions. *Numerische Mathematik*, 1970.

[33] Greengard, S. Advertising gets personal. *Communications of the ACM*, 55(8): 18, August 2012. ISSN 00010782. doi: 10.1145/2240236.2240243.

[34] Gu, M., Zha, H., Ding, C., He, X., Simon, H., and Xia, J. Spectral relaxation models and structure analysis for k-way graph clustering and bi-clustering. Technical report, 2001.

[35] Hansen, L. Bayesian averaging is well-temperated. *Advances in Neural Information Processing Systems*, pages 265–271, 1999.

[36] Hartigan, J. A. Direct Clustering of a Data Matrix. *Journal of the American Statistical Association*, 67(337):123–129, March 1972. ISSN 0162-1459. doi: 10.1080/01621459.1972.10481214.

[37] Hastie, T., Tibshirani, R., and Friedman, J. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York, New York, NY, 2009. ISBN 978-0-387-84857-0. doi: 10.1007/b94608.

[38] Hastings, W. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 1970.

[39] Herlau, T., Schmidt, M. N., and Mø rup, M. Infinite-degree-corrected stochastic block model. *Physical Review E*, 90(3):032819, September 2014. ISSN 1539-3755. doi: 10.1103/PhysRevE.90.032819.

[40] Holland, P. W. and Leinhardt, S. An Exponential Family of Probability Distributions for Directed Graphs. *Journal of the American Statistical Association*, 76:33–50, 1981. ISSN 0162-1459. doi: 10.2307/2287037.

[41] Holland, P. W., Laskey, K. B., and Leinhardt, S. Stochastic blockmodels: First steps. *Social Networks*, 5(2):109–137, June 1983. ISSN 03788733. doi: 10.1016/0378-8733(83)90021-7.

[42] Hu, Y. J. Performance-based Pricing Models in Online Advertising. *SSRN Electronic Journal*, 2004. ISSN 1556-5068. doi: 10.2139/ssrn.501082.

[43] IAB and PwC. IAB internet advertising revenue report, 2013 full year report. *Interactive Advertising Bureau (IAB) and PricewaterhouseCoopers (PwC). http://goo.gl/NwAIzr [Last visit 2014-11-29]*, 2013.

[44] Ishwaran, H. and James, L. F. Gibbs Sampling Methods for Stick-Breaking Priors. *Journal of the American Statistical Association*, 96(453):161–173, March 2001. ISSN 0162-1459. doi: 10.1198/016214501750332758.

[45] Jansen, B. J. and Mullen, T. Sponsored search: an overview of the concept, history, and technology. *International Journal of Electronic Business*, 6(2):114, 2008. ISSN 1470-6067. doi: 10.1504/IJEB.2008.018068.

[46] Karrer, B. and Newman, M. E. J. Stochastic blockmodels and community structure in networks. *Physical Review E*, 83(1):016107, January 2011. ISSN 1539-3755. doi: 10.1103/PhysRevE.83.016107.

[47] Katehakis, M. N. and Veinott, A. F. J. The multi-armed bandit problem: decomposition and computation. *Mathematics of Operations Research*, 12(2):262–268, 1987.

[48] Kaufman, L. and Rousseeuw, P. Clustering by means of medoids. *Statistical Data Analysis Based on the L1–Norm and Related Methods*, 1987.

[49] Kemp, C., Tenenbaum, J. B., Griffiths, T. L., Yamada, T., and Ueda, N. Learning Systems of Concepts with an Infinite Relational Model. In *Proceedings of the 21st National Conference on Artificial Intelligence*, pages 381–388, 2006. ISBN 978-1-57735-281-5.

[50] Latouche, P., Birmelé, E., and Ambroise, C. Overlapping stochastic block models with application to the French political blogosphere. *The Annals of Applied Statistics*, 5(1):309–336, March 2011. ISSN 1932-6157. doi: 10.1214/10-AOAS382.

[51] Lee, D. and Seung, H. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999.

[52] McMahan, H. Follow-the-regularized-leader and mirror descent: Equivalence theorems and l1 regularization. *International Conference on Artificial Intelligence and Statistics*, 2011.

[53] McMahan, H., Holt, G., and Sculley, D. Ad click prediction: a view from the trenches. *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2013.

[54] Menon, A. A. K. and Elkan, C. A log-linear model with latent features for dyadic prediction. *2010 IEEE International Conference on Data Mining*, pages 364–373, December 2010. doi: 10.1109/ICDM.2010.148.

[55] Menon, A. A. K., Chitrapura, K.-P. K., Garg, S., Agarwal, D., and Kota, N. Response prediction using collaborative filtering with hierarchies and side-information. *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '11*, page 141, 2011. doi: 10.1145/2020408.2020436.

[56] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

[57] Miller, K., Jordan, M., and Griffiths, T. Nonparametric latent feature models for link prediction. *Advances in neural information processing systems*, 9:1276–1284, 2009.

[58] Mirkin, B. Mathematical Classification and Clustering. *Journal of the Operational Research Society*, 48(8):852–852, August 1997. ISSN 0160-5682. doi: 10.1057/palgrave.jors.2600836.

[59] Morup, M., Schmidt, M. N., and Hansen, L. K. Infinite multiple membership relational modeling for complex networks. *Machine Learning for Signal Processing (MLSP), 2011 IEEE International Workshop on*, 2011.

[60] Neal, R. Probabilistic inference using Markov chain Monte Carlo methods. *Technical Report CRG-TR-93-1. Department of Computer Science, University of Toronto, Canada*, 1993.

[61] Ng, A., Jordan, M., and Weiss, Y. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 2:849–856, 2002.

[62] Nocedal, J. and Wright, S. J. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer-Verlag, New York, 1999. ISBN 0-387-98793-2. doi: 10.1007/b98874.

[63] Nvidia. GPU Computing: The Revolution. *http://www.nvidia.com/object/cuda_home_new.html*, (Date accessed 11/21/2014), 2014.

[64] Palla, G., Derényi, I., Farkas, I., and Vicsek, T. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 2005.

[65] Pitman, J. Exchangeable and partially exchangeable random partitions. *Probability Theory and Related Fields*, 102:145–158, 1995. ISSN 01788051. doi: 10.1007/BF01213386.

[66] Robbins, H. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 1952.

[67] Solove, D. J. *The Digital Person: Technology and Privacy in the Information Age*. NYU Press, 2004. ISBN 0814798462.

[68] Spielmat, D. and Teng, S. Spectral partitioning works: Planar graphs and finite element meshes. *Foundations of Computer Science, Proceedings, 37th Annual Symposium on. IEEE*, 1996.

[69] Srebro, N. Maximum-margin matrix factorization. *Advances in neural information processing systems*, pages 1329–1336, 2004.

[70] Tsuruoka, Y., Tsujii, J., and Ananiadou, S. Stochastic gradient descent training for L1-regularized log-linear models with cumulative penalty. *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, 1:477, 2009. doi: 10.3115/1687878.1687946.

[71] Wahlgreen, B. O. and Hansen, L. K. Large scale topic modeling made practical. *2011 IEEE International Workshop on Machine Learning for Signal Processing*, 1(11):1–6, September 2011. doi: 10.1109/MLSP.2011.6064628.

[72] White, H. C., Boorman, S. A., and Breiger, R. L. Social Structure from Multiple Networks. I. Blockmodels of Roles and Positions, 1976. ISSN 0002-9602.

[73] Whittle, P. Multi-armed bandits and the Gittins index. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 143–149, 1980.

[74] Xu, W., Liu, X., and Gong, Y. Document clustering based on non-negative matrix factorization. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, volume pages, page 273. ACM, 2003.

[75] Zou, H. and Hastie, T. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 2005.

{2.7182818284}